ARDUINO

65 PRAKTYCZNYCH PROJEKTÓW

JOHN BOXALL

Helion

Tytuł oryginału: Arduino Workshop: A Hands-On Introduction with 65 Projects
Tłumaczenie: Mikołaj Szczepaniak
ISBN: 978-83-246-8002-3

Original edition Copyright © 2013 by John Boxall.
All rights reserved.

Published by arrangement with No Starch Press, Inc.

Polish edition copyright © 2014 by Helion SA.
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/ardupp.zip

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/ardupp_ebook
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- Poleć książkę na Facebook.com
- Kup w wersji papierowej
- Oceń książkę

- Księgarnia internetowa
- Lubię to! » Nasza społeczność

Książkę dedykuję dwóm osobom, które zawsze we mnie wierzyły:
mojej mamie i mojej najdroższej Kathleen

Spis treści

PODZIĘKOWANIA **17**

1
WPROWADZENIE **19**
Nieograniczone możliwości 20
Popularność 24
Części i akcesoria 24
Wymagane oprogramowanie 25
System Mac OS X 25
System Windows XP i nowsze 29
System Ubuntu Linux 9.04 i nowsze 33
Bezpieczeństwo 36
Co dalej? 36

2
PIERWSZE SPOJRZENIE NA PŁYTKĘ ARDUINO I ŚRODOWISKO IDE **37**
Płytka Arduino 37
Wprowadzenie do środowiska IDE 42
Obszar poleceń 43
Obszar tekstu 44
Obszar komunikatów 44
Tworzenie pierwszego szkicu w środowisku IDE 45
Komentarze 45
Funkcja setup() 46
Sterowanie sprzętem 46
Funkcja loop() 47
Weryfikacja szkicu 49
Wysyłanie i uruchamianie szkicu 50
Modyfikowanie szkicu 50
Co dalej? 50

3
PIERWSZE KROKI 51
Planowanie projektów 52
Kilka słów o elektryczności 53
Natężenie 53
Napięcie 53
Moc 53
Komponenty elektroniczne 53
Rezystor 54
Dioda LED 57
Płytka uniwersalna 58
Projekt nr 1: tworzenie fali migających diod LED 61
Algorytm 61
Sprzęt 61
Szkic 61
Schemat 62
Uruchamianie szkicu 63
Stosowanie zmiennych 64
Projekt nr 2: powtarzanie instrukcji za pomocą pętli for 65
Zmiana jasności diod LED za pomocą modulacji szerokości impulsu 66
Projekt nr 3: przykład zastosowania metody PWM 67
Dodatkowe komponenty elektroniczne 68
Tranzystor 68
Dioda prostownicza 69
Przekaźnik 70
Obwody zasilane wyższym napięciem 71
Co dalej? 72

4
ELEMENTY SKŁADOWE OBWODÓW 73
Stosowanie schematów obwodów 74
Identyfikacja komponentów 75
Połączenia na schemacie obwodu 77
Analiza przykładowego schematu 77
Kondensator 78
Mierzenie pojemności kondensatora 78
Odczytywanie wartości kondensatorów 79
Rodzaje kondensatorów 79
Wejście cyfrowe 80
Projekt nr 4: przykład użycia cyfrowego wejścia 82
Algorytm 82
Sprzęt 83
Schemat obwodu 83
Szkic 87
Modyfikowanie szkicu 87

Wyjaśnienie szkicu 87
Tworzenie stałych za pomocą wyrażenia #define 88
Odczytywanie stanu pinów cyfrowych 88
Podejmowanie decyzji za pomocą wyrażenia if 88
Podejmowanie dodatkowych decyzji za pomocą wyrażeń if-then-else 89
Zmienne logiczne 90
Operatory porównania 90
Łączenie wielu operacji porównania 91
Projekt nr 5: sterowanie ruchem samochodowym 92
Cel 92
Algorytm 93
Sprzęt 93
Schemat 93
Szkic 94
Uruchamianie szkicu 97
Sygnały analogowe kontra sygnały cyfrowe 98
Projekt nr 6: tester baterii 99
Cel 99
Algorytm 99
Sprzęt 100
Schemat 100
Szkic 100
Działania arytmetyczne w systemie Arduino 102
Zmienne typu float 102
Operatory porównania liczb 103
Poprawa precyzji pomiarów sygnału analogowego za pomocą napięcia referencyjnego 103
Stosowanie zewnętrznego napięcia referencyjnego 103
Stosowanie wewnętrznego napięcia referencyjnego 104
Rezystor nastawny 105
Brzęczyki piezoelektryczne 106
Schemat elementu piezo 107
Projekt nr 7: praktyczne wykorzystanie brzęczyka piezo 107
Projekt nr 8: budowa szybkiego termometru 108
Cel 109
Sprzęt 109
Schemat 110
Szkic 110
Doskonalenie szkicu 112
Co dalej? 112

5
PRACA Z FUNKCJAMI 113
Projekt nr 9: tworzenie funkcji powtarzającej określone działanie 114
Projekt nr 10: tworzenie funkcji ustawiającej liczbę cykli włączania diod 115
Tworzenie funkcji zwracającej wartość 116

Projekt nr 11: budowa szybkiego termometru z migającymi diodami LED 117
Sprzęt 117
Schemat 118
Szkic 118
Wyświetlanie danych odbieranych od płytki Arduino w oknie monitora portu szeregowego 120
Monitor portu szeregowego 120
Projekt nr 12: wyświetlanie temperatury w oknie monitora portu szeregowego 122
Diagnozowanie systemów za pomocą monitora portu szeregowego 123
Podejmowanie decyzji za pomocą wyrażeń while 124
Konstrukcja do-while 125
Wysyłanie danych z monitora portu szeregowego do systemu Arduino 125
Projekt nr 13: mnożenie liczby przez dwa 126
Zmienne typu long 127
Projekt nr 14: stosowanie zmiennych typu long 128
Co dalej? 129

6
LICZBY, ZMIENNE I DZIAŁANIA ARYTMETYCZNE 131
Generowanie liczb losowych 132
Generowanie liczb losowych na podstawie napięcia na wolnym pinie 132
Projekt nr 15: tworzenie elektronicznej kostki do gry 134
Sprzęt 134
Schemat 134
Szkic 134
Modyfikowanie szkicu 137
Krótkie wprowadzenie w świat liczb binarnych 137
Zmienne typu byte 137
Zwiększanie liczby dostępnych pinów cyfrowych za pomocą rejestrów przesuwających 138
Projekt nr 16: tworzenie wyświetlacza liczb binarnych złożonego z diod LED 140
Sprzęt 140
Łączenie rejestru przesuwającego 74HC595 140
Szkic 142
Projekt nr 17: implementacja binarnego quizu 143
Algorytm 143
Szkic 143
Tablice 146
Definiowanie tablicy 146
Odwoływanie się do wartości w tablicy 147
Zapisywanie i odczytywanie danych przechowywanych w tablicach 147
Siedmiosegmentowe wyświetlacze LED 148
Sterowanie wyświetlaczem LED 150

Projekt nr 18: tworzenie wyświetlacza jednocyfrowego 151
Sprzęt 151
Schemat 151
Szkic 151
Wyświetlanie dwóch cyfr 153
Projekt nr 19: sterowanie dwoma modułami wyświetlaczy siedmiosegmentowych LED 154
Sprzęt 154
Schemat 154
Modulo 155
Projekt nr 20: budowa termometru cyfrowego 156
Sprzęt 157
Szkic 157
Moduły wyświetlaczy matrycowych LED 158
Schemat obwodu wyświetlacza matrycowego LED 159
Łączenie obwodu 161
Arytmetyka bitowa 161
Operator koniunkcji bitowej 162
Operator alternatywy bitowej 162
Operator bitowej alternatywy wykluczającej 163
Operator negacji bitowej 163
Operatory bitowego przesunięcia w lewo i w prawo 163
Projekt nr 21: tworzenie wyświetlacza matrycowego LED 164
Projekt nr 22: wyświetlanie obrazów na wyświetlaczu matrycowym LED 165
Projekt nr 23: wyświetlanie obrazu na wyświetlaczu matrycowym LED 167
Projekt nr 24: prezentacja animacji na wyświetlaczu matrycowym LED 169
Szkic 169
Co dalej? 170

7
WYŚWIETLACZE CIEKŁOKRYSTALICZNE 171
Znakowe moduły LCD 172
Obsługa znakowego modułu LCD w szkicu 173
Wyświetlanie tekstu 174
Wyświetlanie zmiennych i liczb 175
Projekt nr 25: definiowanie znaków niestandardowych 176
Graficzne moduły LCD 178
Łączenie graficznego modułu LCD 179
Stosowanie modułu LCD 179
Sterowanie wyświetlaczem 180
Projekt nr 26: funkcje tekstowe w praktyce 180
Tworzenie złożonych efektów wizualnych 181
Projekt nr 27: budowa szybkiego termometru z wyświetlaną historią 183
Algorytm 183
Sprzęt 183

Szkic ... 184
Wynik ... 185
Modyfikowanie szkicu ... 186
Co dalej? ... 186

8
ROZSZERZANIE MOŻLIWOŚCI PLATFORMY ARDUINO ... 187
Moduły ... 188
Płytka prototypowa ProtoShield ... 190
Projekt nr 28: tworzenie modułu niestandardowego z ośmioma diodami LED ... 191
Sprzęt ... 192
Schemat ... 192
Układ płytki prototypowej ProtoShield ... 192
Projekt ... 193
Lutowanie komponentów ... 194
Modyfikacja modułu niestandardowego ... 195
Rozszerzanie szkiców za pomocą bibliotek ... 196
Importowanie bibliotek dla modułów ... 196
Karty pamięci microSD ... 201
Testowanie karty microSD ... 201
Projekt nr 29: zapisywanie danych na karcie pamięci ... 202
Projekt nr 30: budowa urządzenia rejestrującego temperaturę ... 205
Sprzęt ... 205
Szkic ... 205
Zarządzanie czasem wykonywania aplikacji za pomocą funkcji millis() i micros() ... 208
Projekt nr 31: budowa stopera ... 210
Sprzęt ... 210
Schemat obwodu ... 210
Szkic ... 210
Przerwania ... 213
Tryby przerwań ... 213
Konfiguracja przerwań ... 214
Aktywowanie i dezaktywowanie przerwań ... 214
Projekt nr 32: stosowanie przerwań ... 214
Szkic ... 214
Co dalej? ... 216

9
KLAWIATURY NUMERYCZNE ... 217
Stosowanie klawiatury numerycznej ... 217
Łączenie klawiatury numerycznej ... 218
Programowanie obsługi klawiatury numerycznej ... 219
Testowanie szkicu ... 220
Podejmowanie decyzji za pomocą konstrukcji switch-case ... 220

Projekt nr 33: tworzenie zamka sterowanego klawiaturą numeryczną 221
Szkic 221
Działanie szkicu 223
Testowanie szkicu 223
Co dalej? 224

10
ODCZYTYWANIE DANYCH WEJŚCIOWYCH UŻYTKOWNIKA ZA POŚREDNICTWEM EKRANÓW DOTYKOWYCH 225
Ekrany dotykowe 226
Łączenie ekranu dotykowego 226
Projekt nr 34: adresowanie obszarów na ekranie dotykowym 227
Sprzęt 227
Szkic 227
Testowanie szkicu 229
Odwzorowywanie punktów dotknięcia ekranu 229
Projekt nr 35: budowa dwustanowego przełącznika dotykowego 230
Szkic 231
Działanie szkicu 232
Testowanie szkicu 233
Projekt nr 36: budowa przełącznika dotykowego podzielonego na trzy obszary 233
Mapa ekranu dotykowego 233
Szkic 234
Działanie szkicu 235
Co dalej? 236

11
RODZINA PRODUKTÓW ARDUINO 237
Projekt nr 37: budowa własnej platformy Arduino na płytce uniwersalnej 238
Sprzęt 238
Schemat obwodu 241
Uruchamianie szkicu testowego 244
Bogata rodzina płytek Arduino 247
Płytka Arduino Uno 249
Płytka Freetronics Eleven 249
Płytka Freeduino 250
Płytka Boarduino 250
Płytka Arduino Nano 251
Płytka Arduino LilyPad 251
Płytka Arduino Mega 2560 252
Płytka Freetronics EtherMega 253
Płytka Arduino Due 253
Co dalej? 254

12
SILNIKI I RUCH 255
Wprawianie urządzeń w ruch za pomocą silników wykonawczych 256
Wybór silnika wykonawczego 256
Łączenie silnika wykonawczego 257
Uruchamianie silnika wykonawczego 257
Projekt nr 38: budowa termometru analogowego 259
Sprzęt 259
Schemat 259
Szkic 260
Stosowanie silników elektrycznych 261
Tranzystor Darlingtona TIP120 262
Projekt nr 39: sterowanie silnikiem 262
Sprzęt 262
Schemat 263
Szkic 264
Projekt nr 40: budowa robota gąsienicowego i sterowanie tym robotem 265
Sprzęt 265
Schemat 267
Szkic 270
Wykrywanie kolizji 272
Projekt nr 41: wykrywanie kolizji robota za pomocą mikroprzełącznika 272
Schemat 273
Szkic 273
Czujniki odległości na podczerwień 276
Łączenie obwodu 276
Testowanie czujnika odległości na podczerwień 276
Projekt nr 42: wykrywanie kolizji robota za pomocą czujnika odległości na podczerwień 279
Ultradźwiękowe czujniki odległości 281
Łączenie czujnika ultradźwiękowego 282
Stosowanie czujnika ultradźwiękowego 282
Testowanie ultradźwiękowego czujnika odległości 282
Projekt nr 43: wykrywanie kolizji robota za pomocą ultradźwiękowego czujnika odległości 284
Szkic 284
Co dalej? 287

13
STOSOWANIE SYSTEMU GPS NA PLATFORMIE ARDUINO 289
Czym jest GPS? 290
Testowanie modułu GPS 291
Projekt nr 44: budowa prostego odbiornika GPS 293
Sprzęt 293
Szkic 294
Wyświetlanie położenia na ekranie LCD 295

Projekt nr 45: budowa precyzyjnego zegara korzystającego z systemu GPS 296
Sprzęt 296
Szkic 296
Projekt nr 46: rejestrowanie położenia ruchomego obiektu w czasie 298
Sprzęt 298
Szkic 298
Wyświetlanie zarejestrowanych lokalizacji na mapie 300
Co dalej? 302

14
BEZPRZEWODOWE PRZESYŁANIE DANYCH 303
Stosowanie niedrogich modułów komunikacji bezprzewodowej 304
Projekt nr 47: zdalne, bezprzewodowe sterowanie urządzeniem 305
Sprzęt składający się na obwód nadajnika 305
Schemat nadajnika 306
Sprzęt składający się na obwód odbiornika 306
Schemat odbiornika 306
Szkic nadajnika 308
Szkic odbiornika 309
Moduły bezprzewodowego przesyłania danych XBee
— większy zasięg i szybsza transmisja 310
Projekt nr 48: transmisja danych za pomocą modułów XBee 312
Szkic 312
Konfiguracja komputera pod kątem odbierania danych 313
Projekt nr 49: budowa zdalnie sterowanego termometru 314
Sprzęt 314
Układ urządzenia 315
Szkic 315
Obsługa urządzenia 317
Co dalej? 317

15
ZDALNE STEROWANIE ZA POMOCĄ PODCZERWIENI 319
Czym jest podczerwień? 319
Przygotowanie do użycia podczerwieni 320
Odbiornik podczerwieni 320
Pilot 321
Szkic testowy 321
Testowanie układu 322
Projekt nr 50: zdalne sterowanie systemem Arduino na podczerwień 323
Sprzęt 323
Szkic 323
Rozszerzanie szkicu 325

Projekt nr 51: budowa zdalnie sterowanego robota 325
Sprzęt 325
Szkic 326
Co dalej? 328

16
ODCZYTYWANIE ETYKIET RFID 329
Wewnętrzna budowa urządzeń RFID 330
Testowanie sprzętu 331
Schemat 331
Testowanie schematu 331
Projekt nr 52: budowa prostego systemu kontroli dostępu na bazie technologii RFID 333
Szkic 333
Działanie szkicu 335
Zapisywanie danych we wbudowanej pamięci EEPROM systemu Arduino 336
Odczytywanie i zapisywanie danych w pamięci EEPROM 337
Projekt nr 53: budowa systemu kontroli dostępu RFID z pamięcią „ostatniej akcji" 338
Szkic 338
Działanie szkicu 341
Co dalej? 341

17
MAGISTRALE DANYCH 343
Magistrala I^2C 344
Projekt nr 54: stosowanie zewnętrznej pamięci EEPROM 346
Sprzęt 346
Schemat 346
Szkic 347
Wynik 349
Projekt nr 55: stosowanie układu ekspandera portów 350
Sprzęt 350
Schemat 350
Szkic 352
Magistrala SPI 353
Łączenie pinów 353
Implementacja obsługi magistrali SPI 354
Wysyłanie danych do urządzenia SPI 355
Projekt nr 56: stosowanie cyfrowego rezystora nastawnego 356
Sprzęt 356
Schemat 357
Szkic 357
Co dalej? 359

18
ZEGARY CZASU RZECZYWISTEGO 361
Łączenie modułu RTC 362
Projekt nr 57: wyświetlanie daty i godziny na podstawie zegara czasu rzeczywistego 362
Sprzęt 363
Szkic 363
Działanie szkicu 365
Projekt nr 58: tworzenie prostego zegara cyfrowego 367
Sprzęt 368
Szkic 368
Działanie szkicu i generowane wyniki 371
Projekt nr 59: budowa systemu kontroli czasu pracy na bazie technologii RFID 371
Sprzęt 372
Szkic 373
Działanie szkicu 377
Co dalej? 377

19
INTERNET 379
Czego potrzebujemy? 379
Projekt nr 60: budowa zdalnej stacji monitoringu 381
Sprzęt 381
Szkic 381
Rozwiązywanie problemów 384
Działanie szkicu 385
Projekt nr 61: ćwierkające Arduino 386
Sprzęt 386
Szkic 386
Sterowanie systemem Arduino za pośrednictwem strony internetowej 388
Projekt nr 62: konfiguracja usługi Teleduino i zdalne sterowanie systemem Arduino 389
Sprzęt 389
Szkic 389
Zdalne sterowanie systemem Arduino 391
Co dalej? 392

20
KOMUNIKACJA W SIECI TELEFONII KOMÓRKOWEJ 393
Sprzęt 394
Przygotowanie modułu zasilania 395
Konfigurowanie i testowanie sprzętu 396
Zmiana częstotliwości pracy 398

Projekt nr 63: budowa dzwoniącego Arduino 400
Sprzęt 400
Schemat 401
Szkic 401
Działanie szkicu 402
Projekt nr 64: budowa systemu Arduino wysyłającego SMS-y 403
Szkic 403
Działanie szkicu 404
Projekt nr 65: konfiguracja systemu sterowanego za pomocą wiadomości SMS 405
Sprzęt 405
Schemat 405
Szkic 405
Działanie szkicu 408
Co dalej? 408

SKOROWIDZ 411

Podziękowania

Chciałbym najpierw wyrazić ogromną wdzięczność zespołowi pracującemu nad rozwojem Arduino: Massimowi Banziemu, Davidowi Cuartiellesowi, Tomowi Igoe, Gianluce Martinowi i Davidowi Mellisowi. Bez Waszej wizji, ale też ciężkiej pracy to wielkie dzieło nie byłoby możliwe.

Dziękuję także mojemu recenzentowi, Marcowi Alexandrowi, za jego zaangażowanie, wiedzę, sugestie, wsparcie, pomysły i długie rozmowy, a także za wielką wytrwałość, bezcenną podczas realizacji tak wielkiego projektu.

Chciałbym też podziękować następującym organizacjom za udostępnienie obrazów i okazaną pomoc: adafruit industries, Agilent Technologies, Gravitech, Freetronics, Oomlout, Seeed Studio, Sharp Corporation i SparkFun. Jestem winien szczególne podziękowania firmie Freetronics, której doskonałe produkty umożliwiły realizację opisanych tutaj projektów. Dziękuję również wszystkim programistom, którzy poświęcili swój wolny czas na tworzenie bibliotek dla platformy Arduino — to dzięki ich zaangażowaniu praca konstruktorów jest dużo prostsza.

Jestem też pod wrażeniem pracy zespołu rozwijającego projekt Fritzing, czyli doskonałe narzędzie open source do projektowania obwodów, które wykorzystywałem niemal na każdym kroku podczas pisania tej książki.

Dziękuję także następującym osobom (kolejność nazwisk jest przypadkowa) za zachęty do pracy, wsparcie, inspirację: Iraphne Childs, Limor Fried, Jonathanowi Oxerowi, Philipowi Lindsayowi, Nicole Kilah, Kenowi Shirriffowi, Nathanowi Kennedy'emu, Davidowi Jonesowi oraz Nathanowi Seidle'owi.

I wreszcie dziękuję wszystkim pracownikom wydawnictwa No Starch Press: Sondrze Silverhawk za pomysł napisania tej książki, Serenie Yang za pomoc redakcyjną, nieskończoną cierpliwość i bezcenne sugestie, Billowi Pollockowi za wsparcie i rady oraz za konsekwentne sugerowanie alternatywnych, lepszych sposobów tłumaczenia pewnych zagadnień.

1

Wprowadzenie

Niemal każdemu z nas zdarzają się sytuacje, w których przygląda się jakiemuś gadżetowi i zastanawia się, jak to urządzenie *naprawdę* działa? Może to być zdalnie sterowany model łodzi, system sterujący pracą windy, automat sprzedający (i zliczający wrzucane monety) lub jakaś elektroniczna zabawka. Wielu z nas idzie krok dalej i marzy o budowie własnego robota lub układu elektronicznego na potrzeby kolejki elektrycznej lub w celu zbierania i analizowania danych o pogodzie. Od czego należy zacząć?

Płytka Arduino (pokazana na rysunku 1.1) może znacznie ułatwić odkrywanie tajemnic urządzeń elektronicznych poprzez praktyczne eksperymenty. System Arduino, którego oryginalną wersję zaprojektowali Massimo Banzi i David Cuartielles, oferuje możliwość budowy interaktywnych projektów (zdalnie sterowanych robotów, systemów śledzenia pozycji według sygnału GPS czy gier elektronicznych) za stosunkowo niewysoką cenę.

Projekt Arduino jest dynamicznie rozwijany od momentu wprowadzenia na rynek w 2005 roku. Wokół tego projektu powstał dobrze prosperujący przemysł stale wspierany przez osoby, których cechą wspólną jest dążenie do tworzenia nowych rozwiązań. Społeczność konstruktorów obejmuje zarówno osoby prywatne, jak i większe grupy, w tym organizacje komercyjne, kluby hobbystów oraz instytucje edukacyjne — wszystkie te osoby i podmioty są zainteresowane eksperymentami z systemem Arduino.

Rysunek 1.1. Płytka Arduino

Aby docenić różnorodność projektów realizowanych na bazie systemu Arduino, wystarczy przeszukać internet. Każdy może bez trudu znaleźć listy grup udostępniających programy i kursy wprowadzające w świat systemu Arduino — wszystkie te materiały są przygotowywane przez kreatywnych hobbystów.

Nieograniczone możliwości

Wystarczy pobieżnie przejrzeć tę książkę, aby zrozumieć, że płytka Arduino może służyć zarówno do realizacji tak prostych zadań jak miganie niewielkim źródłem światła, jak i do bardziej złożonych zastosowań, jak komunikacja z telefonem komórkowym. Istnieje też mnóstwo rozwiązań pośrednich o przeciętnym poziomie złożoności.

Warto na przykład zapoznać się z urządzeniem zaprojektowanym przez Philipa Lindsaya (patrz rysunek 1.2). Wiadomości tekstowe można wysyłać do urządzenia za pośrednictwem telefonu komórkowego. Komunikaty mogą być wyświetlane na przykład w salach tanecznych. Urządzenie używa płytki Arduino i modułu telefonu komórkowego do otrzymywania wiadomości tekstowych wysyłanych z innych telefonów (podobnie jak w projekcie nr 65). Komunikaty tekstowe są następnie przekazywane do pary dużych, niedrogich wyświetlaczy z matrycami punktowymi.

Rysunek 1.2. Tekst wiadomości SMS na wyświetlaczu z matrycą punktową

Duże wyświetlacze, które można dość łatwo łączyć z systemem Arduino, są dostępne w przystępnych cenach, zatem nie ma potrzeby budowy samego wyświetlacza od podstaw. (Więcej informacji na temat tego projektu można znaleźć na stronie *http://www.labradoc.com/i/follower/p/project-sms-text-scroller*).

Czy system Arduino można wykorzystać do zorganizowania niepowtarzalnych oświadczyn? Tyler Cooper chciał w oryginalny sposób oświadczyć się swojej dziewczynie i zbudował w tym celu urządzenie nazwane „urządzeniem do odwróconego geocachingu" — czyli niewielkie pudełko zawierające pierścionek zaręczynowy (patrz rysunek 1.3). Po przeniesieniu pudełka w odpowiednie miejsce (określane na podstawie sygnału GPS) zamek otwierał się, udostępniając romantyczną wiadomość i sam pierścionek. Podobne urządzenie można łatwo zbudować samemu za pomocą płytki Arduino, odbiornika GPS, modułu LCD (na przykład takiego jak w rozdziale 13.) oraz niewielkiego silnika wykonawczego, który będzie zabezpieczał zawartość pudełka do momentu jego przeniesienia we właściwe miejsce. Także niezbędny kod źródłowy jest dość prosty — odpowiednie rozwiązanie można zaimplementować w zaledwie kilka godzin. Najbardziej czasochłonnym zadaniem jest wybór odpowiedniego pudełka, w którym zmieści się cały system. (Więcej informacji na temat tego projektu można znaleźć na stronie *http://learn.adafruit.com/reverse-geocache-engagement-box/*).

Rysunek 1.3. Oryginalne oświadczyny przy użyciu systemu Arduino

A oto inny ciekawy przykład. Kurt Schulz chciał monitorować poziom naładowania baterii w swoim skuterze elektrycznym. Kiedy jednak odkrył, jak prosta jest praca z systemem Arduino, znacznie rozwinął swój projekt — w ten sposób powstał kompleksowy system zarządzania skuterem elektrycznym nazwany Scooterputer. System Scooterputer nie tylko mierzy poziom napięcia baterii, ale też wyświetla bieżącą prędkość, przebyty odcinek, kąt nachylenia pokonywanego wzniesienia, temperaturę, godzinę, datę, pozycję GPS itp. System obejmuje też moduł telefonu komputerowego, dzięki czemu może być używany zdalnie zarówno do śledzenia trasy, jak i wyłączenia silnika (w razie kradzieży skutera).

Całym systemem można sterować za pomocą niewielkiego ekranu dotykowego (patrz rysunek 1.4). Każdą funkcję tego systemu można traktować jako osobny element składowy. Podobny system można zbudować w zaledwie kilka weekendów. (Więcej informacji o projekcie można znaleźć na stronie *http://www.janspace.com/b2evolution/arduino.php/2010/06/26/scooterputer/*).

Rysunek 1.4. Wyświetlacz systemu Scooterputer (dzięki uprzejmości Kurta Schulza)

I wreszcie warto wspomnieć o Johnie Sariku, miłośniku popularnej łamigłówki Sudoku i cyfrowych lamp jarzeniowych (wyświetlaczy Nixie). John połączył obie pasje i zbudował wielki, 81-cyfrowy komputer do gry w Sudoku! Użytkownik może rozgrywać pełną grę na planszy 9×9, a system Arduino zarządza rozgrywką i sprawdza poprawność podawanych wartości. Mimo że projekt w tej formie wielu początkującym konstruktorom może się wydać zaawansowany, w rzeczywistości jest możliwy do zrealizowania przez każdego, a zastosowane układy elektroniczne nie są zbyt skomplikowane. Gotowe urządzenie jest dość duże i doskonale prezentuje się na ścianie (patrz rysunek 1.5). (Więcej informacji o projekcie można znaleźć na stronie *http://trashbearlabs.wordpress.com/2010/07/09/nixie-sudoku/*).

System Arduino został użyty także do budowy urządzenia TwypeWriter przez firmę Oomlout. W urządzeniu wykorzystano płytkę Arduino z modułem sieciowym (ang. *Ethernet shield*) podłączonym do internetu w celu przeszukiwania Twittera pod kątem określonych słów kluczowych. W momencie znalezienie wystąpienia jednego z tych słów kluczowych odpowiedni wpis jest wysyłany do elektronicznej maszyny do pisania w celu wydrukowania. Płytka Arduino jest połączona z układem klawiatury tej maszyny, dzięki czemu może emulować proces wpisywania tekstu przez prawdziwego użytkownika (patrz rysunek 1.6). (Więcej informacji na temat projektu można znaleźć na stronie *http://oomlout.co.uk/blog/twitter-monitoring-typewritter-twypwriter/*).

Rysunek 1.5. Plansza do Sudoku złożona z cyfrowych lamp jarzeniowych

Rysunek 1.6. Urządzenie TwypeWriter

To tylko kilka przypadkowo wybranych przykładów ilustrujących możliwości oferowane przez system Arduino. Każdy z nas może łatwo tworzyć własne projekty — po przeczytaniu tej książki żaden, nawet najbardziej zaawansowany projekt nie powinien sprawić większego kłopotu.

Popularność

Popularność platformy Arduino rośnie każdego dnia. Czytelnicy, którym najłatwiej przychodzi nauka w szerszym gronie, mogą poszukać w internecie stron dla hasła *Cult of Arduino*, aby poznać osiągnięcia innych użytkowników i trafić do grup pracujących nad projektami na bazie platformy Arduino. Członkowie tych grup nierzadko są prawdziwymi artystami, których pomysłowość podczas budowy projektów bywa naprawdę zaskakująca. Niektórzy z nich próbują nawet tworzyć własne małe płytki zgodne z Arduino. Współpraca z tymi grupami może być ciekawym doświadczeniem, pozwoli poznać interesujących ludzi i umożliwi dzielenie się własną wiedzą na temat systemu Arduino z innymi hobbystami.

Części i akcesoria

Jak każde urządzenie elektroniczne, płytka Arduino jest dostępna w wielu sklepach detalicznych, które oferują szeroką gamę produktów i akcesoriów. Przed dokonaniem zakupu warto się upewnić, że przedmiotem transakcji rzeczywiście jest oryginalna płytka Arduino, nie jedna z jej podróbek. W przeciwnym razie urządzenie najprawdopodobniej nie będzie działało prawidłowo lub będzie dużo mniej wydajne. W dłuższej perspektywie budowa projektów na bazie tańszych zamienników może się okazać droższa niż w przypadku stosowania oryginalnych komponentów. Listę dostawców płytki Arduino można znaleźć na stronie *http://arduino.cc/en/Main/Buy/*.

Poniższa lista zawiera spis aktualnie działających sklepów (w kolejności alfabetycznej), które sam mogę polecić jako dostawców części i akcesoriów dla platformy Arduino:

- Adafruit Industries (*http://www.adafruit.com/*)
- DigiKey (*http://www.digikey.com/*)
- Jameco Electronics (*http://www.jameco.com/*)
- Little Bird Electronics (*http://www.littlebirdelectronics.com/*)
- Newark (*http://www.newark.com/*)
- nicegear (*http://www.nicegear.co.nz/*)
- Oomlout (*http://www.oomlout.co.uk/*)
- RadioShack (*http://www.radioshack.com/*)
- RS Components (*http://www.rs-components.com/*)
- SparkFun Electronics (*http://www.sparkfun.com/*)

Jak łatwo stwierdzić podczas lektury dalszej części tej książki, posługuję się wieloma produktami zgodnymi z platformą Arduino zakupionymi w sklepie Freetronics (*http://www.freetronics.com/*). Wszystkie niezbędne komponenty są jednak dość popularne, zatem można je bez trudu znaleźć także w innych sklepach.

Nie należy jednak spieszyć się z zakupami. Zachęcam najpierw do przeczytania pierwszych kilku rozdziałów, które pozwolą lepiej zrozumieć, co naprawdę będzie potrzebne do realizacji własnych projektów — w ten sposób można uniknąć nieprzemyślanego kupowania zbędnych produktów.

Wymagane oprogramowanie

Do programowania systemu Arduino wystarczy dowolny komputer z zainstalowanym oprogramowaniem, które zwykle określa się mianem zintegrowanego środowiska programowania (ang. *integrated development environment — IDE*). Warunkiem uruchomienia tego oprogramowania jest dysponowanie jednym z następujących systemów operacyjnych:

- Mac OS X lub nowszy;
- Windows XP (32- lub 64-bitowy) lub nowszy;
- Linux (32- lub 64-bitowy), dystrybucja Ubuntu lub podobna.

Warto teraz pobrać i zainstalować to środowisko IDE, zatem powinieneś od razu przejść do punktu poświęconego właściwemu systemowi operacyjnemu i postępować zgodnie z instrukcjami w tym punkcie. Warto też sprawdzić, czy dysponujesz przewodem USB potrzebnym do połączenia platformy Arduino z komputerem, i — w razie jego braku — dokupić odpowiedni przewód. Nawet użytkownicy, którzy nie mają jeszcze płytki Arduino, mogą bez trudu pobrać środowisko IDE i zapoznać się z jego możliwościami. Ponieważ numery wersji środowiska IDE są dość często zmieniane, wersje opisane w tej książce mogą być inne niż numery bieżących wersji, jednak same procedury instalacji powinny pozostać identyczne.

UWAGA *Już po przekazaniu oryginalnego wydania tej książki do druku wystąpiły pewne problemy związane z instalacją w systemie Windows 8. Użytkownicy systemu Windows 8 powinni odwiedzić forum poświęcone platformie Arduino (http://arduino.cc/forum/index.php/topic,94651.15.html), na którym wyjaśniono problem i zasugerowano odpowiednie rozwiązania.*

System Mac OS X

W tym podpunkcie opisano procedurę pobierania i konfiguracji środowiska Arduino IDE w systemie Mac OS X.

Instalacja środowiska IDE

Aby zainstalować środowisko IDE w systemie Mac, należy wykonać następujące kroki:

1. W przeglądarce internetowej (na przykład Safari) należy otworzyć stronę udostępniającą wersję środowiska IDE do pobrania pod adresem *http://arduino.cc/en/Main/Software/* (patrz rysunek 1.7).

Rysunek 1.7. Strona udostępniająca środowisko IDE do pobrania otwarta w przeglądarce Safari

2. Należy kliknąć link *Mac OS X*. Rozpocznie się pobieranie pliku. Postęp tego procesu i komunikat o pobraniu pliku będą widoczne w oknie *Pobieranie* (patrz rysunek 1.8).

Rysunek 1.8. Informacja o zakończeniu pobierania pliku

3. Po zakończeniu pobierania należy dwukrotnie kliknąć plik, aby rozpocząć proces instalacji. Na ekranie zostanie wyświetlone okno podobne do tego z rysunku 1.9.

Rysunek 1.9. Nowy folder środowiska Arduino IDE

UWAGA *Plik reprezentowany przez trzecią ikonę na rysunku 1.9 wymaga instalacji tylko w sytuacji, gdy stosowana płytka Arduino jest starsza od obecnej wersji Uno.*

4. Należy przeciągnąć ikonę *Arduino* nad folder *Applications* i zwolnić przycisk myszy. W trakcie kopiowania tego pliku zostanie wyświetlone tymczasowe okno ze stanem tego procesu.

5. Należy teraz podłączyć płytkę Arduino do komputera Mac za pomocą przewodu USB. Po krótkiej chwili zostanie wyświetlone okno dialogowe podobne do tego z rysunku 1.10.

Rysunek 1.10. Komunikat o wykryciu płytki Arduino. W niektórych przypadkach komunikat może wskazywać urządzenie Uno (zamiast Eleven)

6. Należy teraz kliknąć przycisk *Network Preferences...*, a następnie przycisk *Apply* (w wyświetlonym oknie *Network*). Komunikat o statusie *not configured* można zignorować.

Konfiguracja środowiska IDE

Po pobraniu środowiska IDE należy wykonać następujące kroki, aby otworzyć i skonfigurować to środowisko:

1. Należy otworzyć folder *Programy* w programie Finder (patrz rysunek 1.11), a następnie dwukrotnie kliknąć ikonę *Arduino.app*.

Rysunek 1.11. Folder Applications

2. Na ekranie może się pojawić okno z ostrzeżeniem o otwieraniu aplikacji pobranej z internetu. W takim przypadku należy kliknąć przycisk *Otwórz*, aby kontynuować. Na ekranie zostanie wyświetlone zainstalowane środowisko IDE (patrz rysunek 1.12).

Rysunek 1.12. Środowisko IDE w systemie Mac OS X

3. Proces konfiguracji jest już prawie zakończony — do przygotowania środowiska Arduino IDE do pracy wystarczą już tylko dwa kroki. Po pierwsze, należy określić, do którego gniazda podłączono płytkę Arduino. W tym celu należy wybrać opcję */dev/tty.usbmodem1d11* z menu *Narzędzia/Port szeregowy* (patrz rysunek 1.13).

Rysunek 1.13. Wybór portu USB

4. Ostatni krok polega na określeniu, którą wersję płytki Arduino podłączono do komputera. To bardzo ważne, ponieważ różnice dzielące poszczególne modele są dość istotne. Użytkownicy dysponujący najbardziej popularnym modelem Uno powinni wybrać opcję *Narzędzia/Płytka/Arduino Uno* (patrz rysunek 1.14). Różnice dzielące poszczególne płytki Arduino zostaną szczegółowo wyjaśnione w rozdziale 11.

Sprzęt i oprogramowanie są teraz gotowe do pracy. Użytkownicy systemu Mac OS X mogą przejść do podrozdziału „Bezpieczeństwo” na końcu tego rozdziału.

Rysunek 1.14. Wybór właściwej wersji płytki Arduino

System Windows XP i nowsze

W tym podpunkcie opisano procedurę pobierania oprogramowania, instalacji sterownika i konfiguracji środowiska IDE w systemie Windows.

Instalacja środowiska IDE

Aby zainstalować środowisko Arduino IDE w systemie Windows, należy wykonać następujące kroki:

1. W przeglądarce internetowej (na przykład Firefox) należy otworzyć stronę udostępniającą wersję środowiska IDE do pobrania pod adresem *http://arduino.cc/en/Main/Software/* (patrz rysunek 1.15).
2. Należy kliknąć link *Windows (ZIP file)*. Przeglądarka wyświetli okno dialogowe podobne do tego z rysunku 1.16. Należy wybrać opcję *Otwórz za pomocą Eksplorator Windows*, a następnie kliknąć przycisk *OK*. Postęp procesu pobierania pliku można wyświetlić, klikając ikonę *Pokaż okno pobierania plików* (patrz rysunek 1.17).
3. Po zakończeniu pobierania należy dwukrotnie kliknąć plik — na ekranie zostanie wyświetlone okno podobne do tego z rysunku 1.18.

Rysunek 1.15. Strona udostępniająca środowisko IDE do pobrania otwarta w przeglądarce Firefox

Rysunek 1.16. Pobieranie pliku

Rysunek 1.17. Przeglądarka Firefox wyświetlająca postęp pobierania pliku

Rysunek 1.18. Pakiet środowiska IDE

4. Folder nazwany *arduino-1.0.5* (lub podobny, zależnie od wersji) należy skopiować w miejsce przechowywania aplikacji. Po skopiowaniu folderu należy przejść do miejsca docelowego i znaleźć ikonę aplikacji Arduino (patrz rysunek 1.19). Aby ułatwić dostęp do aplikacji w przyszłości, warto umieścić skrót do pliku aplikacji na pulpicie systemu Windows.

Rysunek 1.19. Folder środowiska IDE z zaznaczoną ikoną aplikacji Arduino

Instalacja sterowników

Następne zadanie polega na instalacji sterowników interfejsu USB płytki Arduino.

1. Należy teraz podłączyć płytkę Arduino do komputera PC za pomocą przewodu USB. Po krótkiej chwili zostanie wyświetlony komunikat o błędzie (z informacją *Sterownik urządzenia nie został poprawnie zainstalowany* lub podobną). Wystarczy zamknąć to okno dialogowe lub chmurkę.

2. Należy otworzyć Panel sterowania systemu Windows. Po otwarciu Menedżera urządzeń należy przewinąć listę w dół i znaleźć pozycję reprezentującą płytkę Arduino (patrz rysunek 1.20).

Rysunek 1.20. Menedżer urządzeń

3. Należy kliknąć element *Arduino Uno* w sekcji Inne urządzenia i wybrać z menu kontekstowego opcję *Aktualizuj oprogramowanie sterownika*. W wyświetlonym oknie dialogowym należy wybrać opcję *Przeglądaj mój komputer w poszukiwaniu oprogramowania sterownika*. Po wyświetleniu następnego okna dialogowego należy kliknąć przycisk *Przeglądaj* i przejść do folderu *drivers* w nowo zainstalowanym folderze środowiska Arduino IDE (patrz rysunek 1.21). Należy teraz kliknąć przycisk *OK*.

Rysunek 1.21. Lokalizacja folderu drivers

4. W następnym oknie dialogowym należy kliknąć przycisk *Dalej*. System Windows może teraz wyświetlić komunikat o braku możliwości weryfikacji wydawcy oprogramowania sterownika. W takim przypadku należy kliknąć przycisk *Mimo to instaluj ten sterownik*. Po krótkiej chwili system Windows poinformuje użytkownika o zakończeniu instalacji sterownika i wyświetli numer portu COM, do którego podłączono płytkę Arduino (patrz rysunek 1.22).

Rysunek 1.22. Komunikat o prawidłowej aktualizacji sterowników

Konfiguracja środowiska IDE

Proces instalacji i konfiguracji jest już prawie zakończony — do zakończenia konfiguracji środowiska Arduino IDE wystarczą dwa kroki.

1. Należy otworzyć środowisko Arduino IDE. Użytkownik musi wskazać środowisku IDE rodzaj gniazda, do którego podłączono płytkę Arduino — w tym celu należy kliknąć port *Narzędzia/Port szeregowy*, a następnie wybrać numer portu COM zgodnie z komunikatem wyświetlonym podczas aktualizacji sterownika.
2. Ostatni krok polega na określeniu, którą wersję płytki Arduino podłączono do komputera. To o tyle ważne, że różne modele płytki Arduino zasadniczo różnią się od siebie. Użytkownicy dysponujący Uno powinni wybrać opcję *Narzędzia/Płytka/Arduino Uno*. Różnice dzielące poszczególne płytki Arduino zostaną szczegółowo wyjaśnione w rozdziale 11.

Po skonfigurowaniu środowiska Arduino IDE można przejść do podrozdziału „Bezpieczeństwo” w dalszej części tego rozdziału.

System Ubuntu Linux 9.04 i nowsze

Poniższe instrukcje przygotowano z myślą o użytkownikach systemu Ubuntu Linux, którzy chcą pobrać i skonfigurować środowisko Arduino IDE.

Instalacja środowiska IDE

Aby zainstalować środowisko IDE, należy wykonać następujące kroki:

1. W przeglądarce internetowej (na przykład Firefox) trzeba otworzyć stronę udostępniającą wersję środowiska IDE do pobrania pod adresem *http://arduino.cc/en/Main/Software/* (patrz rysunek 1.23).
2. W zależności od stosowanego systemu należy kliknąć link *32-bit* lub *64-bit* obok słowa *Linux*. Po wyświetleniu okna dialogowego podobnego do tego z rysunku 1.24 należy wybrać opcję *Otwórz za pomocą Menedżer archiwów* i kliknąć przycisk *OK*.

Rysunek 1.23. Strona udostępniająca środowisko IDE do pobrania otwarta w przeglądarce Firefox w systemie Ubuntu

Rysunek 1.24. Pobieranie pliku

3. Po pobraniu pliku archiwum zostanie wyświetlone w programie Archive Manager (patrz rysunek 1.25). Folder nazwany *arduino-1.0.5* (lub podobny, zależnie od wersji) należy skopiować w miejsce przechowywania innych aplikacji lub do folderu domowego.

Rysunek 1.25. Pakiet środowiska IDE

Konfiguracja środowiska IDE

Należy jeszcze skonfigurować środowisko IDE.

1. W pierwszym kroku należy podłączyć płytkę Arduino do komputera PC za pomocą przewodu USB. Ponieważ w tym kroku trzeba uruchomić środowisko Arduino IDE, należy znaleźć skopiowany wcześniej folder *arduino-1.0.5* i dwukrotnie kliknąć plik *arduino* (patrz rysunek 1.26).

Rysunek 1.26. Folder środowiska IDE z zaznaczonym plikiem arduino

2. Jeśli zostanie wyświetlone okno dialogowe podobne do tego z rysunku 1.27, należy kliknąć przycisk *Uruchom*, aby uruchomić środowisko IDE (patrz rysunek 1.28).

Rysunek 1.27. Nadawanie uprawnień uruchamiania środowiska IDE

Rysunek 1.28. Środowisko IDE w systemie Ubuntu

3. Po uruchomieniu środowiska IDE należy jeszcze określić rodzaj gniazda, do którego podłączono płytkę Arduino. Z menu *Narzędzia/Port szeregowy* należy wybrać opcję */dev/ttyACMx*, gdzie *x* to pojedyncza cyfra (zwykle jest dostępny tylko jeden port z taką nazwą).
4. Należy następnie określić, którą wersję płytki Arduino podłączono do komputera. To o tyle ważne, że różne modele płytki Arduino zasadniczo różnią się od siebie. Użytkownicy dysponujący Uno powinni wybrać opcję *Narzędzia/Płytka/Arduino Uno*. Różnice dzielące poszczególne płytki Arduino zostaną szczegółowo wyjaśnione w rozdziale 11.

Sprzęt i oprogramowanie są teraz gotowe do pracy.

Bezpieczeństwo

Jak przy każdym hobby czy pracy, także w tym przypadku tylko od nas zależy bezpieczeństwo nasze i osób w naszym otoczeniu. W tej książce będę omawiał metody realizacji projektów przy użyciu prostych narzędzi ręcznych, urządzeń elektrycznych zasilanych bateriami, ostrych nożyków i przecinaków, a w niektórych przypadkach także lutownicy. Żaden z projektów nie będzie wymagał bezpośredniej ingerencji w urządzenia zasilane prądem o napięciu sieciowym. Zadania tego rodzaju należy pozostawić elektrykom dysponującym odpowiednimi uprawnieniami i doświadczeniem. Trzeba pamiętać, że kontakt z prądem sieciowym może doprowadzić do śmierci.

Co dalej?

Przed nami mnóstwo zabawy i ciekawa przygoda — będziemy tworzyli rzeczy, które większości z nas do tej pory wydawały się niemożliwe do zbudowania. W tej książce opiszę 65 projektów na bazie płytki Arduino, od bardzo prostych do dość skomplikowanych. Wszystkie te rozwiązania zaprojektowano z myślą o ułatwianiu konkretnych czynności. Do dzieła!

2

Pierwsze spojrzenie na płytkę Arduino i środowisko IDE

W tym rozdziale przyjrzymy się bliżej płytce Arduino oraz oprogramowaniu (środowisku IDE), które będzie służyło do wysyłania szkiców (ang. *sketch*; tak są nazywane programy pisane dla tej platformy) na samą płytkę Arduino. Omówię podstawowy szkielet szkicu oraz kilka prostych funkcji, które można implementować w ramach szkiców. Na końcu tego rozdziału stworzymy i wyślemy pierwszy wspólny szkic.

Płytka Arduino

Czym dokładnie jest Arduino? Zgodnie z definicją na oficjalnej stronie internetowej Arduino (*http://www.arduino.cc/*):

> Arduino jest platformą open source do tworzenia prototypów elektronicznych zbudowaną na bazie elastycznego, łatwego w użyciu sprzętu i oprogramowania. Platformę stworzono z myślą o artystach, projektantach, hobbystach i wszystkich, którzy są zainteresowani budową interaktywnych obiektów lub środowisk.

Mówiąc najprościej, Arduino jest miniaturowym systemem komputerowym, który można programować za pomocą odpowiednich instrukcji w celu operowania na danych wejściowych i wyjściowych w najróżniejszych formach. Dostępny obecnie model płytki Arduino (nazwany Uno) jest stosunkowo mały — dużo mniejszy niż przeciętna dłoń dorosłego człowieka (patrz rysunek 2.1).

Rysunek 2.1. Płytka Arduino Uno jest dość mała

System Arduino, mimo że na pierwszy rzut oka sprawia dość niepozorne wrażenie, umożliwia tworzenie urządzeń zdolnych do interakcji ze światem zewnętrznym. Dzięki możliwości łączenia tego systemu z niemal wszystkimi dostępnymi urządzeniami wejściowymi i wyjściowymi (czujnikami, wskaźnikami, wyświetlaczami, silnikami itp.) konstruktor może zaprogramować wszystkie interakcje niezbędne do stworzenia funkcjonalnego rozwiązania. Na przykład artyści wykorzystali tę platformę do budowy instalacji z migającymi światłami reagującymi na ruch przechodniów. Uczniowie szkoły średniej zbudowali autonomiczny robot wykrywający i gaszący płomienie. Geografowie zaprojektowali systemy monitorowania temperatury i wilgotności (gromadzone pomiary były wysyłane do biura w formie wiadomości tekstowych). Wystarczy poświęcić chwilę na przeszukanie internetu, aby znaleźć mnóstwo innych, równie zaskakujących przykładów.

Przejdźmy teraz do bliższej analizy *sprzętu* składającego się na płytkę Arduino Uno (tzw. elementów fizycznych tego systemu). W razie problemów ze zrozumieniem przeznaczenia opisywanych tutaj elementów nie ma powodu do zmartwień, ponieważ wszystkie te komponenty zostaną bardziej szczegółowo omówione w dalszych rozdziałach tej książki.

Zacznijmy od przeglądu najważniejszych komponentów na płytce Uno. Z lewej strony płytki są dostępne dwa złącza (patrz rysunek 2.2).

Rysunek 2.2. Port USB i złącze zasilania

Z lewej strony zamontowano złącze USB (od ang. *universal serial bus*). Złącze to spełnia trzy ważne zadania: zapewnia zasilanie płytki, wysyła instrukcje do systemu Arduino lub wysyła dane z i do komputera. Po prawej stronie zainstalowano złącze zasilania. Za pośrednictwem tego złącza można zasilać płytkę Arduino, stosując standardowy zasilacz sieciowy.

W środkowej części (nieco bliżej dolnej krawędzi) zainstalowano serce płytki, czyli mikrokontroler (patrz rysunek 2.3).

Rysunek 2.3. Mikrokontroler

Mikrokontroler jest „mózgiem" platformy Arduino. To miniaturowy komputer obejmujący procesor (odpowiedzialny za wykonywanie rozkazów), zawierający różne typy pamięci (w których są przechowywane dane i instrukcje zapisane w szkicach) oraz obsługujący wiele kanałów wysyłania i otrzymywania danych. Bezpośrednio pod mikrokontrolerem znajdują się dwa wiersze małych gniazd (patrz rysunek 2.4).

Pierwszy wiersz udostępnia połączenia zasilania i umożliwia podłączenie zewnętrznego przycisku *RESET*. Drugi wiersz oferuje sześć wejść analogowych przydatnych podczas mierzenia sygnałów elektrycznych ze zmieniającym się napięciem. Co więcej, piny A4 i A5 mogą być używane także do wysyłania danych do innych urządzeń i odbierania danych od tych urządzeń. Przy górnej krawędzi płytki zamontowano dwa dodatkowe wiersze gniazd (patrz rysunek 2.5).

Rysunek 2.4. Gniazda zasilania i analogowe

Rysunek 2.5. Piny wejść i wyjść cyfrowych

Gniazda oznaczone etykietami od 0 do 13 pełnią funkcję pinów wejścia-wyjścia (I/O). Za pośrednictwem tych pinów można albo wykrywać sygnały elektryczne, albo generować te sygnały na potrzeby innych urządzeń. Piny oznaczone numerami 0 i 1 dodatkowo pełnią funkcję portu szeregowego, który umożliwia wysyłanie danych do innych urządzeń i odbieranie danych od tych urządzeń (podobnie jak w przypadku złącza USB używanego do komunikacji z komputerem). Piny oznaczone znakiem tyldy (~) mogą dodatkowo generować sygnały elektryczne o różnych napięciach, które są przydatne na przykład podczas generowania efektów świetlnych lub sterowania silnikami elektrycznymi.

Na płytce można też znaleźć bardzo przydatne elementy nazywane *diodami LED* (od ang. *light-emitting diodes*). Te niewielkie urządzenia świecą, gdy przepływa przez nie prąd. Na płytce Arduino zainstalowano cztery diody LED: jedną z prawej strony (oznaczoną etykietą ON i wskazującą zasilanie całej płytki) oraz trzy w odrębnej grupie (patrz rysunek 2.6).

Rysunek 2.6. Diody LED na płytce Arduino

Diody LED oznaczone etykietami *TX* i *RX* świecą odpowiednio w czasie wysyłania i odbierania danych przez system Arduino za pośrednictwem portu szeregowego i złącza USB. Dioda LED oznaczona literą *L* jest dostępna dla programisty (jest połączona z pinem cyfrowym wejścia-wyjścia nr 13). Niewielki czarny prostokąt na lewo od diod LED to mikrokontroler sterujący interfejsem USB — to dzięki niemu płytka Arduino może wysyłać dane do komputera i odbierać dane z komputera bez potrzeby każdorazowego implementowania rozwiązań niskopoziomowych.

I wreszcie na rysunku 2.7 pokazano przycisk *RESET*.

Rysunek 2.7. Przycisk RESET

Tak jak w przypadku tradycyjnych komputerów, także w systemie Arduino czasem coś nie idzie po naszej myśli — jeśli zawodzą wszystkie inne rozwiązania, należy zresetować i ponownie uruchomić system Arduino. Przycisk *RESET* na płytce Arduino (patrz rysunek 2.7) służy właśnie do ponownego uruchamiania systemu w celu rozwiązywania poważnych problemów.

Jedną z największych zalet systemu Arduino jest łatwość jego rozszerzania, czyli dodawania fizycznych urządzeń wprowadzających nowe funkcje. Dwa wiersze gniazd po obu stronach płytki Arduino umożliwiają podłączanie do systemu tzw. *modułów*, czyli dodatkowych płytek z odpowiednio rozmierzonymi pinami. Na przykład na rysunku 2.8 pokazano moduł zawierający interfejs sieci internetowej (Ethernet), dzięki któremu system Arduino może się komunikować za pośrednictwem sieci lokalnej lub internetu. Moduł dodatkowo oferuje sporo miejsca dla ewentualnych niestandardowych obwodów.

Rysunek 2.8. Moduł sieciowy zainstalowany na płytce Arduino

Warto zwrócić uwagę na wiersze gniazd dostępne także na tym module. Dzięki nim istnieje możliwość dodawania kolejnych modułów ponad modułem sieciowym. Na rysunku 2.9 pokazano przykładową strukturę obejmującą dodatkowy moduł z dużym wyświetlaczem, czujnikiem temperatury, dodatkową pamięcią oraz wielką diodą LED.

Rysunek 2.9. Moduł z wyświetlaczem cyfrowym i czujnikiem temperatury

Warto podkreślić, że to do użytkownika systemu należy zwracanie uwagi na wykorzystanie poszczególnych pinów wejściowych i wyjściowych przez instalowane moduły, tak aby nie dochodziło do konfliktów pomiędzy tymi urządzeniami. Istnieje też możliwość dokupienia zupełnie pustych płytek, na których każdy może budować własne obwody. Do tego zagadnienia wrócimy w rozdziale 8.

Ważnym uzupełnieniem dla komponentów sprzętowych składających się na platformę Arduino jest *oprogramowanie*, czyli zbiór instrukcji określających, co i jak ten sprzęt ma robić. Oprogramowanie dzieli się na dwa typy: pierwszym z nich jest zintegrowane środowisko programowania (IDE), które zostanie omówione w dalszej części tego rozdziału; drugim typem są szkice platformy Arduino tworzone przez samych użytkowników.

Środowisko IDE, które należy zainstalować na zwykłym komputerze, służy do przygotowywania szkiców i wysyłania ich na płytkę Arduino.

Wprowadzenie do środowiska IDE

Środowisko programowania Arduino IDE przypomina prosty edytor tekstu (patrz rysunek 2.10). Okno środowiska zostało podzielone na trzy główne obszary: obszar poleceń, obszar tekstu i obszar komunikatów.

Rysunek 2.10. Środowisko programowania Arduino IDE

Obszar poleceń

Widoczny na rysunku 2.10 obszar poleceń obejmuje pasek tytułu, elementy menu oraz ikony. Pasek tytułu wyświetla nazwę pliku szkicu (w tym przypadku *sketch_jun06a*) oraz wersję samego środowiska IDE (w tym przypadku *Arduino 1.0.5*). Pod paskiem tytułu są dostępne elementy menu (*Plik*, *Edycja*, *Szkic*, *Narzędzia* i *Pomoc*) oraz ikony — poszczególne elementy tych sekcji zostaną opisane poniżej.

Elementy menu

Tak jak w przypadku edytorów tekstu, także w środowisku Arduino IDE użytkownik może kliknąć jeden z elementów menu, aby wyświetlić dostępne opcje.

Menu ***Plik*** zawiera opcje umożliwiające zapisywanie, ładowanie i drukowanie szkiców, bogaty zbiór przykładowych szkiców dostępnych do otwarcia oraz ważną opcję *Preferencje*.

Menu ***Edycja*** obejmuje standardowe funkcje kopiowania, wklejania i przeszukiwania (znane z edytorów tekstu).

Menu ***Szkic*** zawiera polecenia weryfikacji szkicu przed wysłaniem na płytkę oraz opcje folderów szkiców i importowania bibliotek.

Menu ***Narzędzia*** zawiera różne funkcje i polecenia umożliwiające wybór typu płytki Arduino oraz portu USB.

Menu ***Pomoc*** zawiera linki do rozmaitych tematów ułatwiających pracę ze środowiskiem IDE oraz opcję wyświetlania bieżącej wersji tego środowiska.

Ikony

Pod menu znajduje się sześć ikon. Umieszczenie kursora myszy nad każdą z tych ikon powoduje wyświetlenie jej nazwy. Poszczególne ikony (od lewej strony) opisano poniżej:

Weryfikuj — kliknięcie tej ikony powoduje sprawdzenie, czy szkic systemu Arduino jest prawidłowy i nie zawiera błędów programistycznych.

Załaduj — kliknięcie tej ikony powoduje sprawdzenie szkicu i wysłanie go na płytkę Arduino.

Nowy — kliknięcie tej ikony powoduje otwarcie nowego, pustego szkicu w nowym oknie.

Otwórz — ta ikona umożliwia otwarcie zapisanego szkicu.

Zapisz — kliknięcie tej ikony powoduje zapisanie otwartego szkicu. Jeśli wcześniej nie określono nazwy tego szkicu, użytkownik zostanie poproszony o jej podanie.

Monitor portu szeregowego — kliknięcie tej ikony powoduje otwarcie nowego okna prezentującego dane wysyłane pomiędzy płytką Arduino a środowiskiem IDE.

Obszar tekstu

Obszar tekstu (widoczny w środkowej części zrzutu z rysunku 2.10) jest w istocie miejscem tworzenia szkiców. Nazwa bieżącego szkicu jest wyświetlana w zakładce w lewym górnym rogu obszaru tekstu. (Domyślna nazwa zawiera bieżącą datę). Kod szkicu można wpisywać dokładnie tak, jak wpisuje się tekst w edytorze tekstu.

Obszar komunikatów

Obszar komunikatów jest widoczny w dolnej części rysunku 2.10. Komunikaty środowiska Arduino IDE są wyświetlane właśnie w tym czarnym polu. Widoczne komunikaty są uzależnione od bieżącej sytuacji i obejmują informacje o weryfikacji szkiców, aktualizacjach stanu środowiska itp.

W prawym dolnym rogu obszaru komunikatów jest wyświetlana nazwa stosowanej płytki Arduino oraz numer portu USB, do którego ta płytka została podłączona (w tym przypadku odpowiednio *Arduino Uno* i *COM1*).

Tworzenie pierwszego szkicu w środowisku IDE

Szkic systemu Arduino ma postać zbioru instrukcji przygotowanych z myślą o realizacji określonego zadania. Inaczej mówiąc, szkic jest *programem*. W tym podrozdziale zostanie utworzony i wysłany na płytkę Arduino prosty szkic powodujący miganie diody LED na tej płytce (patrz rysunek 2.11). Dioda będzie włączana i wyłączana w 1-sekundowych odstępach.

Rysunek 2.11. Dioda LED na płytce Arduino (obok wielkiej litery L)

UWAGA *Jeśli nie rozumiesz poszczególnych poleceń w ramach szkicu, nie masz powodu do zmartwień. Moim celem w tym podrozdziale jest tylko wykazanie, jak łatwo można spowodować określone działania płytki Arduino — chciałbym, aby ta prosta prezentacja była zachętą do dalszej lektury i poznawania bardziej złożonych rozwiązań.*

W pierwszym kroku należy podłączyć płytkę Arduino do komputera za pomocą przewodu USB. Należy następnie otworzyć środowisko IDE, wybrać opcję *Narzędzia/Port szeregowy* i sprawdzić, czy został wybrany odpowiedni port USB. Dzięki temu użytkownik może być pewien, że płytka Arduino jest prawidłowo podłączona.

Komentarze

Warto zacząć szkic od komentarza, który w przyszłości będzie przypominał, do czego służy ten program. Komentarz to notatka dowolnej długości umieszczona w samym szkicu i napisana z myślą o użytkownikach. Komentarze w szkicach są przydatną formą zamieszczania informacji zarówno dla samego twórcy programu, jak i dla innych programistów — w ten sposób można zapisywać w szkicu instrukcje jego stosowania oraz rozmaite szczegóły na temat działania programu. Podczas programowania płytki Arduino (tworzenia szkiców) warto dodawać komentarze opisujące intencje programisty. Takie komentarze będą cennym źródłem informacji dla samego twórcy programu, jeśli wróci do swojego szkicu po dłuższym czasie.

Aby dodać komentarz zajmujący pojedynczy wiersz, należy umieścić w kodzie dwa prawe ukośniki (//), a następnie właściwy komentarz:

```
// Szkic Blink LED; autor: Mary Smith; data: 2012-09-09
```

Dwa prawe ukośniki informują środowisko IDE, że tekst następujący po tych znakach ma być ignorowany podczas weryfikacji szkicu. (Jak już wspomniałem, weryfikacja szkicu polega na sprawdzeniu przez środowisko IDE, czy kod programu nie zawiera błędów).

Aby umieścić w kodzie komentarz zajmujący co najmniej dwa wiersze, należy wpisać sekwencję znaków /* w wierszu poprzedzającym komentarz oraz sekwencję */ w wierszu następującym po komentarzu:

```
/*
Szkic Arduino Blink LED
autor: Mary Smith; data: 2012-09-09
*/
```

Tak jak dwa prawe ukośniki powodują ignorowanie następującego po nich tekstu, tak symbole /* i */ sprawiają, że środowisko IDE ignoruje tekst pomiędzy nimi.

Po wpisaniu komentarza opisującego przeznaczenie danego szkicu systemu Arduino (w jednej z opisanych powyżej form) należy zapisać ten szkic, wybierając opcję *Plik/Zapisz jako*. Szkicowi należy jeszcze nadać jakąś krótką nazwę (na przykład *blinky*), a następnie kliknąć przycisk *Zapisz*.

Domyślnym rozszerzeniem plików ze szkicami systemu Arduino jest *.ino* — środowisko IDE powinno automatycznie dodać to rozszerzenie do nazwy pliku. Oznacza to, że plik szkicu w tym przypadku zostanie nazwany *blinky.ino* (nowy szkic powinien być widoczny w szkicowniku dostępnym w menu *Plik*).

Funkcja setup()

Następnym krokiem w procesie tworzenia każdego szkicu jest dodanie funkcji `void setup()`. Funkcja zawiera zbiór instrukcji wykonywanych przez system Arduino tylko raz — bezpośrednio po włączeniu lub ponownym uruchomieniu. Aby utworzyć funkcję `setup()`, należy umieścić w szkicu (pod dodanymi wcześniej komentarzami) następujące wiersze:

```
void setup()
{

}
```

Sterowanie sprzętem

Zadaniem programu jest wymuszenie migania diody LED na płytce Arduino. Dioda LED jest połączona z pinem cyfrowym nr 13 platformy Arduino. Pin cyfrowy może służyć albo do wykrywania sygnału elektrycznego, albo do generowania takiego sygnału na podstawie wydawanych poleceń. W tym projekcie będzie generowany sygnał elektryczny włączający diodę LED. Na pierwszy rzut

oka operowanie na sygnałach przekazywanych za pośrednictwem pinów cyfrowych może się wydawać złożone, jednak wszystkie niezbędne techniki zostaną opisane w dalszych rozdziałach. Na razie skoncentrujmy się na tworzeniu pierwszego szkicu.

Należy teraz umieścić w szkicu następujący kod pomiędzy dwoma nawiasami klamrowymi ({ oraz }):

```
pinMode(13, OUTPUT); // ustawia pin cyfrowy nr 13 jako wyjście
```

Liczba 13 na powyższym listingu reprezentuje numer portu cyfrowego, którego dotyczy ta instrukcja. W tym przypadku program włącza tryb danych wyjściowych (`OUTPUT`) dla pinu nr 13, zatem wskazany pin będzie służył do generowania sygnałów elektrycznych. Gdyby pin miał służyć do wykrywania przychodzących sygnałów elektrycznych, należałoby zastosować tryb `INPUT`. Warto zwrócić uwagę na średnik (`;`) na końcu wywołania funkcji `pinMode()`. Każde wywołanie funkcji w szkicach systemu Arduino musi się kończyć średnikiem.

Warto teraz ponownie zapisać szkic, aby nie stracić efektów dotychczasowej pracy.

Funkcja loop()

Musimy pamiętać, że celem tego szkicu jest ciągłe miganie diody LED. Oznacza to, że należy stworzyć funkcję `loop()`, która spowoduje, że system Arduino będzie wielokrotnie wykonywał instrukcje zapisane w szkicu do momentu odłączenia zasilania lub naciśnięcia przycisku *RESET*.

Aby utworzyć pustą funkcję `loop()`, kod wyróżniony pogrubieniem należy umieścić za funkcją `void setup()`. Nową funkcję koniecznie trzeba zakończyć prawym nawiasem klamrowym (}). Uzupełniony szkic warto ponownie zapisać.

```
/*
Szkic Arduino Blink LED
autor: Mary Smith; data: 2012-09-09
*/

void setup()
{
   pinMode(13, OUTPUT); // ustawia pin cyfrowy nr 13 jako wyjście
}
void loop()
{
   // tutaj należy umieścić kod głównej pętli programu
}
```

OSTRZEŻENIE *Środowisko Arduino IDE nie zapisuje szkiców automatycznie, zatem warto możliwie często samemu zapisywać swoją pracę!*

Należy teraz wypełnić funkcję `void loop()` właściwymi instrukcjami, które będą wykonywane przez system Arduino.

W tym celu trzeba umieścić następujący kod pomiędzy nawiasami klamrowymi wyznaczającymi granice tej funkcji. Po wpisaniu kodu warto kliknąć ikonę *Weryfikuj*, aby sprawdzić, czy nowe instrukcje są prawidłowe:

```
digitalWrite(13, HIGH); // włącza pin cyfrowy nr 13
delay(1000); // wstrzymuje działanie na sekundę
digitalWrite(13, LOW); // wyłącza pin cyfrowy nr 13
delay(1000); // wstrzymuje działanie na sekundę
```

Przeanalizujmy teraz poszczególne elementy tego fragmentu kodu. Funkcja `digitalWrite()` steruje napięciem pinu cyfrowego (w tym przypadku pinu nr 13 połączonego z diodą LED). Przekazanie za pośrednictwem drugiego parametru tej funkcji wartości `HIGH` powoduje ustawienie tzw. stanu wysokiego, w którym przez pin przepływa prąd, powodując świecenie diody LED. (Wywołanie tej samej funkcji z parametrem `LOW` spowoduje odcięcie zasilania i — tym samym — wyłączenie diody LED).

Po włączeniu diody LED szkic wstrzymuje działanie na 1 sekundę za pomocą funkcji `delay(1000)`. Funkcja `delay()` wstrzymuje wykonywanie szkicu na określony czas — w tym przypadku na 1000 milisekund, czyli 1 sekundę.

Zasilanie diody LED jest następnie wyłączane za pomocą wywołania funkcji `digitalWrite(13, LOW)`. I wreszcie szkic wstrzymuje działanie na jedną sekundę w czasie, gdy dioda jest wyłączona (ponownie za pomocą wywołania `delay(1000)`).

Kompletny szkic powinien mieć następującą postać:

```
/*
Szkic Arduino Blink LED
autor: Mary Smith; data: 2012-09-09
*/

void setup()
{
   pinMode(13, OUTPUT); // ustawia pin cyfrowy nr 13 jako wyjście
}
void loop()
{
   digitalWrite(13, HIGH); // włącza pin cyfrowy nr 13
   delay(1000); // wstrzymuje działanie na sekundę
   digitalWrite(13, LOW); // wyłącza pin cyfrowy nr 13
   delay(1000); // wstrzymuje działanie na sekundę
}
```

Przed przystąpieniem do dalszych działań koniecznie należy zapisać szkic!

Weryfikacja szkicu

Weryfikacja szkicu pozwala stwierdzić, czy wszystko zostało prawidłowo zapisane, tj. w sposób, który umożliwi interpretację kodu przez system Arduino. Aby sprawdzić poprawność kompletnego szkicu, należy kliknąć ikonę *Weryfikuj* i chwilę poczekać na zakończenie testu. Po sprawdzeniu szkicu w oknie komunikatów powinna zostać wyświetlona stosowna informacja (patrz rysunek 2.12).

Rysunek 2.12. Szkic został prawidłowo zweryfikowany

Komunikat *Kompilowanie zakończone* informuje użytkownika, że szkic jest prawidłowy i może być wysłany na płytkę Arduino. W oknie komunikatów jest też wyświetlana informacja o pamięci zajmowanej przez sprawdzony szkic (w tym przypadku 1076 bajtów). Łączna ilość pamięci dostępnej na platformie Arduino wynosi 32 256 bajtów.

Co stałoby się, gdyby szkic nie był poprawny? Przypuśćmy na przykład, że programista zapomniał o średniku na końcu drugiego wywołania funkcji `delay(1000)`. Jeśli szkic zawiera błąd, kliknięcie ikony *Weryfikuj* spowoduje wyświetlenie w oknie komunikatów informacji o znalezionych usterkach (przykład takiego komunikatu pokazano na rysunku 2.13).

Rysunek 2.13. Okno komunikatów z informacją o błędzie wykrytym podczas weryfikacji kodu

Z komunikatu wynika, że błąd wystąpił w funkcji `void loop()`. Komunikat wskazuje też numer wiersza, który w ocenie środowiska IDE zawiera źródło błędu (zapis `blinky:16` oznacza, że chodzi o 16. wiersz szkicu *blinky*), oraz wyświetla sam błąd (w tym przypadku brakujący średnik: `error: expected ';' before '}'`

`token`). Środowisko IDE powinno też wyróżnić na żółto miejsce znalezienia błędu lub fragment kodu bezpośrednio za tym miejscem. Takie rozwiązanie znacznie ułatwia lokalizację i eliminowanie błędów.

Wysyłanie i uruchamianie szkicu

Po sprawdzeniu, czy kod został wpisany prawidłowo, należy zapisać szkic, upewnić się, że płytka Arduino jest podłączona do komputera, a następnie kliknąć ikonę *Załaduj*. Środowisko IDE być może ponownie zweryfikuje szkic, po czym wyśle sprawdzony program na płytkę Arduino. W trakcie wysyłania diody LED oznaczone etykietami *TX* i *RX* na płytce Arduino (patrz rysunek 2.6) powinny migać, sygnalizując przekazywanie informacji pomiędzy tą płytką a komputerem.

Czas na chwilę prawdy — płytka Arduino powinna teraz wykonywać wysłany szkic. Jeśli kod został przepisany prawidłowo, dioda LED powinna co sekundę zapalać się i gasnąć!

Gratulacje. Właśnie opanowałeś umiejętność wpisywania, weryfikacji i wysyłania szkiców systemu Arduino.

Modyfikowanie szkicu

Już po uruchomieniu szkicu programiści często podejmują decyzję o zmianie jego działania, na przykład poprzez modyfikację czasu oczekiwania pomiędzy włączaniem a wyłączaniem diody LED. Środowisko IDE, które pod wieloma względami przypomina edytor tekstu, umożliwia otwieranie zapisanych wcześniej szkiców, modyfikowanie ich kodu, ponowne zapisywanie szkiców i wysyłanie na płytkę Arduino. Aby na przykład zwiększyć częstotliwość migania diody LED, wystarczy zmienić parametry obu wywołań funkcji `delay()`. Jeśli dioda ma zmieniać stan co jedną czwartą sekundy, należy zmniejszyć wartość tego parametru do 250:

```
delay(250); // wstrzymuje działanie na jedną czwartą sekundy
```

Po wprowadzeniu zmian należy ponownie wysłać szkic na płytkę. Dioda LED powinna teraz migać czterokrotnie szybciej (powinna zapalać się i gasnąć co 250 milisekund).

Co dalej?

Skoro potrafisz już pisać, edytować, zapisywać i wysyłać szkice systemu Arduino, możesz przystąpić do lektury następnego rozdziału, w którym wyjaśnię m.in., jak używać dodatkowych funkcji, jak implementować dobre projekty oraz jak konstruować podstawowe obwody elektroniczne.

3

Pierwsze kroki

W tym rozdziale:

- opanujesz zasady przygotowywania dobrych projektów;
- poznasz podstawowe właściwości prądu elektrycznego;
- zrozumiesz działanie rezystora, diody LED, tranzystora, diody prostowniczej i przekaźnika;
- użyjesz płytki uniwersalnej do budowy obwodów bez konieczności lutowania;
- nauczysz się, jak za pomocą zmiennych całkowitoliczbowych, pętli i cyfrowych pinów wyjściowych tworzyć różne efekty wizualne przy użyciu diody LED.

Czas tchnąć trochę życia w płytkę Arduino. Jak niedługo pokażę, praca z platformą Arduino to coś więcej niż tylko budowanie systemów na bazie tej płytki. W tym rozdziale opiszę, jak planować projekty w taki sposób, aby pomysły konstruktora przybrały rzeczywistą formę. W dalszej części wprowadzę podstawy elektryczności. Znajomość właściwości prądu elektrycznego jest kluczem do zrozumienia tej książki — bez dobrego rozumienia przynajmniej podstaw elektryczności trudno byłoby tworzyć własne projekty. W tym rozdziale opiszę także

komponenty elektroniczne niezbędne do budowy tych projektów. I wreszcie wprowadzę kilka nowych funkcji, które można stosować w roli elementów składowych podczas tworzenia szkiców systemu Arduino.

Planowanie projektów

Podczas prac nad pierwszymi kilkoma projektami często występuje pokusa rozpoczynania realizacji od napisania szkicu implementującego nowy pomysł. Zanim jednak będzie można przystąpić do pisania szkicu, należy wykonać kilka prostych kroków, aby przygotować się do budowy projektu. Płytka Arduino nie czyta w myślach programisty — musi otrzymać precyzyjne instrukcje. Co więcej, nawet jeśli te instrukcje mogą być prawidłowo wykonane przez system Arduino, wynik ich wykonania nie musi być zgodny z oczekiwaniami twórcy (na przykład wskutek przeoczenia jakiegoś niepozornego szczegółu).

Niezależnie od tego, czy tworzony projekt odpowiada tylko za miganie diody, czy ma być zautomatyzowanym modelem kolejki elektrycznej, warunkiem sukcesu jest szczegółowy plan działania.

Podczas projektowania systemów na bazie platformy Arduino należy wykonać następujące kroki:

1. **Zdefiniowanie celu.** Twórca projektu musi określić, co chce osiągnąć.
2. **Napisanie algorytmu.** *Algorytm* to zbiór instrukcji opisujących, jak projekt ma realizować zdefiniowany cel. Algorytm reprezentuje listę kroków niezbędnych do osiągnięcia przyjętego celu.
3. **Wybór sprzętu.** Należy określić, jak komponenty projektu zostaną podłączone do płytki Arduino.
4. **Napisanie szkicu.** Należy utworzyć wstępną wersję programu, który będzie opisywał działanie systemu Arduino.
5. **Łączenie komponentów.** Należy połączyć wszystkie komponenty, przygotować obwody i podłączyć inne potrzebne elementy do płytki Arduino.
6. **Testowanie i diagnozowanie projektu.** Czy system działa? Na tym etapie można zidentyfikować ewentualne błędy i znaleźć ich przyczyny (w szkicu, połączeniach sprzętowych lub algorytmie).

Im więcej czasu twórca projektu poświęci na planowanie działań, tym mniej problemów wystąpi na etapie testowania i diagnozowania projektu.

UWAGA *Nawet w dobrze zaplanowanych projektach występuje czasem zjawisko przerostu funkcji (ang. feature creep). Wspomniane zjawisko ma miejsce wtedy, gdy twórca projektu wpada na nowe pomysły i próbuje je zrealizować w ramach istniejącego projektu, na siłę dodając kolejne elementy do systemu. Jeśli projekt ma być zmieniony, nie warto próbować dodawania lub modyfikowania rozwiązań w istniejącym systemie. Lepszym rozwiązaniem jest rozpoczęcie nowego projektu (w tym ponowne zdefiniowanie jego celu).*

Kilka słów o elektryczności

Warto teraz poświęcić trochę czasu na omówienie podstaw elektryczności, ponieważ już za chwilę przystąpimy do budowy obwodów elektronicznych na bazie płytki Arduino. W największym skrócie elektryczność to forma energii, którą można dość łatwo wykorzystać i przekształcić w ciepło, światło, ruch lub moc. Elektryczność ma trzy główne właściwości, które są bardzo ważne w kontekście budowy projektów: natężenie, napięcie i moc.

Natężenie

Przepływ energii elektrycznej przez obwód określa się mianem *natężenia* (ang. *current*). Natężenie prądu elektrycznego przepływa przez obwód od dodatniej strony źródła zasilania (na przykład baterii) do ujemnej strony źródła zasilania. Prąd o stałym natężeniu i kierunku przepływu określa się mianem *prądu stałego* (ang. *direct current — DC*). W tej książce nie będę poruszał tematu *prądu zmiennego* (ang. *alternating current — AC*). W niektórych obwodach ujemna strona źródła zasilania jest określana mianem *uziemienia* (ang. *ground — GND*). Miarą natężenia jest *amper (A)*. Niewielkie natężenia wyraża się w *miliamperach (mA)*, gdzie 1000 miliamperów to 1 amper.

Napięcie

Napięcie (ang. *voltage*) jest miarą różnicy potencjałów elektrycznych pomiędzy dwoma punktami obwodu elektrycznego (dodatnią i ujemną stroną źródła zasilania). Napięcie elektryczne wyraża się w *woltach (V)*. Im wyższe jest napięcie, tym szybciej prąd przepływa przez obwód.

Moc

Moc jest miarą pracy wykonywanej przez urządzenie zasilane energią elektryczną (konwersji tej energii z jednej postaci na inną). Moc jest wyrażana w *watach (W)*. Na przykład żarówka o mocy 100 W świeci dużo jaśniej od żarówki o mocy 60 W, ponieważ przekształca więcej energii elektrycznej na światło.

Istnieje prosty wzór matematyczny opisujący relację pomiędzy napięciem, natężeniem i mocą:

$$\text{Moc (W)} = \text{Napięcie (V)} \times \text{Natężenie (A)}$$

Komponenty elektroniczne

Skoro wprowadziłem już podstawowe właściwości prądu elektrycznego, czas przejść do omówienia zasilanych w ten sposób komponentów i urządzeń elektronicznych. Komponenty elektroniczne to najróżniejsze części sterujące natężeniem prądu elektrycznego w ramach układu i — tym samym — umożliwiające realizację projektów. Tak jak różne części samochodu współpracują w celu do-

starczenia paliwa, mocy i wprowadzenia pojazdu w ruch, tak *komponenty elektroniczne* współpracują w celu wykorzystania energii elektrycznej, tak aby można było budować przydatne urządzenia.

We wszystkich rozdziałach tej książki będę stopniowo wprowadzał wyspecjalizowane komponenty i wyjaśniał ich działanie. W poniższych punktach opisano tylko wybrane, najważniejsze komponenty.

Rezystor

Niektóre komponenty, na przykład dioda LED na płytce Arduino, wymagają do działania niewielkiego natężenia, zwykle około 10 mA. Zbyt wysokie natężenie prądu przepływającego przez diodę LED powoduje wzrost jej temperatury (nadmiar energii elektrycznej jest przekształcany w energię cieplną), co może prowadzić do przegrzania i zniszczenia diody. Aby ograniczyć natężenie prądu elektrycznego przepływającego przez takie komponenty jak diody LED, należy umieścić rezystor (opornik) pomiędzy źródłem napięcia a zasilanym komponentem. O ile prąd elektryczny przepływa przez miedziany przewód bez przeszkód, o tyle prąd przepływający przez rezystor jest zmniejszsany. Część natężenia jest konwertowana na niewielką ilość energii cieplnej (ilość tej energii jest proporcjonalna do wartości rezystora, tzw. rezystancji, oporności, oporu elektrycznego). Przykłady typowych rezystorów pokazano na rysunku 3.1.

Rysunek 3.1. Typowe rezystory

Rezystancja

Poziom rezystancji (oporu) może być albo stały, albo zmienny. Rezystancję wyraża się w *omach (Ω)*. Wartość rezystancji może wynosić zero omów, tysiące omów (*kiloomy*, *kΩ*) lub nawet miliony omów (*megaomy*, *MΩ*).

Odczytywanie wartości rezystancji

Rezystory są bardzo małe, zatem w większości przypadków nie jest możliwe wydrukowanie wartości rezystancji na samych komponentach. Mimo że oporność można odczytać za pomocą multimetru (miernika uniwersalnego), istnieje też możliwość odczytania tej wartości bezpośrednio z fizycznego rezystora, chociaż nie ma na nim wydrukowanych liczb. Jednym z typowych sposobów wyrażania rezystancji komponentów jest kod reprezentowany przez sekwencję kolorowych pasków. Kod należy odczytywać od lewej strony w następujący sposób:

Pierwszy pasek reprezentuje pierwszą cyfrę wartości rezystancji.

Drugi pasek reprezentuje drugą cyfrę wartości rezystancji.

Trzeci pasek reprezentuje mnożnik (w przypadku rezystorów z czterema paskami) lub trzecią cyfrę wartości rezystancji (w przypadku rezystorów z pięcioma paskami).

Czwarty pasek reprezentuje mnożnik (w przypadku rezystorów z pięcioma paskami).

Piąty pasek reprezentuje tolerancję (klasę dokładności) rezystora.

Kolory pasków na rezystorach wraz z wartościami rezystancji opisano w tabeli 3.1.

Tabela 3.1. Wartości reprezentowane przez paski wydrukowane na powierzchni rezystora (w omach)

Kolor	Omy
Czarny	0
Brązowy	1
Czerwony	2
Pomarańczowy	3
Żółty	4
Zielony	5
Niebieski	6
Fioletowy	7
Szary	8
Biały	9

Piąty pasek reprezentuje tolerancję rezystora. Tolerancja jest miarą klasy dokładności tego komponentu. Ponieważ produkcja rezystorów z precyzyjnie określonymi wartościami rezystancji jest dość trudna, podczas zakupu rezystora można wybrać dopuszczalny margines błędu. Brązowy pasek oznacza 1-procentową tolerancję, złoty pasek oznacza 5-procentową tolerancję, a srebrny pasek — 10-procentową tolerancję.

Przykładowy diagram rezystora pokazano na rysunku 3.2. Żółty, fioletowy i pomarańczowy pasek reprezentują odpowiednio cyfry 4, 7 i 3 (patrz tabela 3.1). Opisana sekwencja pasków reprezentuje wartość 47 000 Ω, czyli 47 kΩ.

Rysunek 3.2. Przykładowy diagram rezystora

Rezystory do montażu powierzchniowego

Rezystory do montażu powierzchniowego (ang. *chip resistors*) nie są oznaczane paskami, tylko kodem liczbowym i literowym (patrz rysunek 3.3). Dwie pierwsze cyfry reprezentują zwykłą wartość rezystancji, natomiast trzecia cyfra reprezentuje liczbę zer następujących po tej wartości. Na przykład rezystor pokazany na rysunku 3.3 ma wartość rezystancji na poziomie 10 000 Ω, czyli 10 kΩ.

Rysunek 3.3. Przykład rezystora do montażu powierzchniowego

UWAGA *Jeśli na niewielkim rezystorze do montażu powierzchniowego występuje liczba i kod literowy (na przykład* 01C*), należy użyć tabel (wystarczy wpisać w wyszukiwarce Google zapytanie* EIA-96 code calculator*).*

Mierniki uniwersalne

Miernik uniwersalny (tzw. *multimetr*) jest wyjątkowo przydatnym i stosunkowo niedrogim urządzeniem testowym umożliwiającym mierzenie między innymi napięcia, oporu elektrycznego i natężenia prądu. Na rysunku 3.4 pokazano przykładowy pomiar wartości rezystora za pomocą multimetru.

Rysunek 3.4. Zmierzona oporność rezystora 560-omowego z 1-procentową tolerancją

Multimetr jest absolutnie niezbędny w przypadku osób dotkniętych ślepotą barw. Jak w przypadku każdego narzędzia, warto zakupić multimetr od sprawdzonego sprzedawcy, zamiast szukać w internecie niepewnych okazji i najtańszych dostępnych ofert.

Moc znamionowa

Moc znamionowa rezystora jest miarą dopuszczalnej mocy (wyrażonej w watach), przy której rezystor działa prawidłowo — przekroczenie tej wartości spowoduje przegrzanie rezystora. Wszystkie rezystory pokazane na rysunku 3.1 mają moc znamionową 0,25 W, czyli moc typową dla rezystorów stosowanych w obwodach na bazie systemu Arduino.

Podczas wyboru rezystora należy przeanalizować relacje łączące moc, natężenie i napięcie. Im wyższe jest natężenie i (lub) napięcie, tym większa musi być moc znamionowa rezystora.

Większa moc znamionowa rezystora zwykle przekłada się na większe wymiary. Na przykład rezystor pokazany na rysunku 3.5 (o mocy znamionowej 5 W) ma 26 mm długości i 7,5 mm szerokości.

Rysunek 3.5. Rezystor o mocy znamionowej 5 W

Dioda LED

Dioda LED jest bardzo popularnym i niezwykle przydatnym komponentem przekształcającym natężenie prądu elektrycznego na światło. Diody LED mają różne kształty, rozmiary i kolory. Przykład typowej diody LED pokazano na rysunku 3.6.

Rysunek 3.6. Czerwona dioda LED o średnicy 5 mm

Podłączanie diod LED do obwodu wymaga ostrożności, ponieważ diody są urządzeniami spolaryzowanymi — oznacza to, że natężenie prądu elektrycznego powinno przechodzić przez diodę tylko w jednym kierunku. Natężenie prądu dociera do diody za pośrednictwem *anody* (strony dodatniej) i opuszcza diodę za pośrednictwem *katody* (strony ujemnej); patrz rysunek 3.7. Próba odwrotnego podłączenia diody i przepływ prądu w przeciwnym kierunku spowoduje uszkodzenie diody LED.

Rysunek 3.7. Przepływ natężenia przez diodę LED

Na szczęście diody LED projektuje się w taki sposób, że określenie, która końcówka jest anodą, a która katodą, nie stanowi żadnego problemu. Nóżka anody jest dłuższa; nóżka katody jest krótsza, a brzeg diody LED od strony katody jest płaski (patrz rysunek 3.8).

Rysunek 3.8. Na podstawie kształtu diody LED można wskazać anodę (dłuższa nóżka) i katodę (płaska część samej diody)

Przed dodaniem diod LED do projektu należy sprawdzić napięcie i natężenie robocze prądu. Typowe czerwone diody LED wymagają napięcia na poziomie około 1,7 V i natężenia od 5 do 20 mA. Wymienione parametry są pewnym utrudnieniem, ponieważ płytka Arduino oferuje zasilanie napięciem 5 V i zapewnia dużo wyższe natężenie. Na szczęście istnieje możliwość zastosowania *rezystora ograniczającego natężenie*, aby zabezpieczyć diodę LED przed przegrzaniem. Jak należy wybrać właściwy rezystor? Wystarczy zastosować prawo Ohma.

Aby obliczyć wartość oporności niezbędnego rezystora ograniczającego natężenie, należy zastosować następujący wzór:

$$R = (U_s - U_f) : I$$

gdzie U_s to napięcie zasilania (w tym przypadku napięcie 5 V zapewniane przez płytkę Arduino); U_f to napięcie przewodzenia diody LED (przyjmijmy, że wynosi 1,7 V); I to natężenie wymagane do działania diody LED (10 mA). (We wzorze należy zastosować wartość wyrażoną w amperach, zatem zamiast 10 mA należałoby zapisać 0,01 A).

Warto teraz spróbować użyć tego wzoru dla wybranych diod LED — przyjmijmy, że U_s wynosi 5 V, U_f wynosi 1,7 V, a I wynosi 0,01 A. Po podstawieniu tych wartości do wzoru wartość R wyniesie 330 Ω. Okazuje się jednak, że dioda LED będzie świeciła także wtedy, gdy natężenie prądu będzie mniejsze niż 10 mA. Zalecaną praktyką jest stosowanie (o ile to możliwe) mniejszych wartości natężenia, aby chronić delikatne komponenty elektroniczne. W tym przypadku dobrym rozwiązaniem będzie użycie rezystora 560 Ω o mocy 0,25 W, który zapewni natężenie na poziomie około 6 mA.

UWAGA *W razie wątpliwości zawsze należy stosować nieco większe rezystory, ponieważ lepiej dysponować diodą LED, która świeci nieco za słabo, niż diodą, która jest spalona!*

Płytka uniwersalna

Prototypowe obwody, które często są zmieniane, wymagają podstawy — elementu przytrzymującego wszystkie komponenty. Najlepszym narzędziem do tego celu jest *płytka uniwersalna* (ang. *broadboard*), która nie wymaga lutowania komponentów. Płytka uniwersalna to plastikowa podstawa z wierszami elektrycznie połączonych gniazd (połączonych tak długo, jak długo płytka nie jest używana w roli deski do krojenia chleba). Płytki uniwersalne są dostępne w różnych wymiarach, kształtach i kolorach (patrz rysunek 3.10).

TRÓJKĄT PRAWA OHMA

Prawo Ohma określa związki łączące natężenie prądu elektrycznego, oporność i napięcie:

napięcie (U) = natężenie (I) × oporność (R)

Oznacza to, że jeśli są znane dwie spośród tych wielkości, obliczenie trzeciej nie stanowi problemu. Popularnym sposobem zapamiętywania prawa Ohma jest przedstawianie tego wzoru w formie trójkąta (patrz rysunek 3.9).

$U = I \times R$

$I = U : R$

$R = U : I$

Rysunek 3.9. Trójkąt prawa Ohma

Trójkątny diagram ilustrujący prawo Ohma jest wygodnym narzędziem obliczania napięcia, natężenia i oporności zawsze wtedy, gdy znane są tylko dwie z tych trzech wartości. Aby na przykład obliczyć oporność, wystarczy zasłonić palcem literę *R* — pozostaną widoczne napięcie (w liczniku) i natężenie (w mianowniku); by obliczyć napięcie, należy zasłonić literę *U* — pozostaną widoczne natężenie i oporność na tym samym poziomie (obie wartości należy pomnożyć).

Rysunek 3.10. Płytki uniwersalne w różnych kształtach i wymiarach

Warunkiem efektywnego korzystania z płytki uniwersalnej jest znajomość sposobu połączenia gniazd — projektant obwodu musi wiedzieć, czy połączenia przebiegają wzdłuż krótszej, czy wzdłuż dłuższej krawędzi. Zdarza się też, że wiersze i kolumny na części płytki są połączone w pionie, a na części w poziomie. Sposób łączenia gniazd zależy od modelu płytki. Na przykład na płytce widocznej

w górnej części rysunku 3.11 kolumny złożone z pięciu otworów są ze sobą połączone w pionie, ale izolowane w poziomie. Jeśli dwa przewody zostaną podłączone do jednego pionowego wiersza, płytka zapewni połączenie elektryczne tych przewodów. Podobnie długie wiersze na środku płytki (pomiędzy rzędami kolumn na górze i dole płytki) są połączone w poziomie. Ponieważ obwody często są łączone ze źródłem zasilania i z uziemieniem, długie poziome wiersze wprost idealnie nadają się właśnie do podłączenia tych dwóch końcówek źródła zasilania.

Rysunek 3.11. Wewnętrzne połączenia na płytce uniwersalnej

W przypadku bardziej złożonych obwodów płytka uniwersalna może się okazać za mała, aby prawidłowo rozmieścić wszystkie komponenty zgodnie z pierwotnymi założeniami. Problem można jednak rozwiązać, stosując specjalne przewody zwierające. Sklepy sprzedające płytki uniwersalne zwykle mają w ofercie także niewielkie opakowania przewodów zwierających różnych długości (przykład takiego zestawu pokazano na rysunku 3.12).

Rysunek 3.12. Przewody zwierające dla płytki uniwersalnej

Projekt nr 1: tworzenie fali migających diod LED

W tym projekcie zostaną użyte diody LED i rezystory. Pięć diod LED będzie emulowało przednie oświetlenie samochodu KITT w popularnym serialu telewizyjnym *Nieustraszony* — czerwone światła będą tworzyły efekt przypominający falę.

Algorytm

Oto algorytm przygotowany dla tego projektu:

1. Włącz diodę LED nr 1.
2. Odczekaj pół sekundy.
3. Wyłącz diodę LED nr 1.
4. Włącz diodę LED nr 2.
5. Odczekaj pół sekundy.
6. Wyłącz diodę LED nr 2.
7. Kontynuuj do momentu włączenia diody LED nr 5, a następnie odwróć ten proces, tak aby były włączane diody LED od 5. do 1.
8. Powtarzaj ten proces w nieskończoność.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- pięć diod LED;
- pięć rezystorów 560 Ω;
- płytka uniwersalna;
- różne przewody;
- płytka Arduino i przewód USB.

Diody LED zostaną podłączone do pinów cyfrowych od 2. do 6. za pośrednictwem rezystorów ograniczających natężenie o wartości oporu równej 560 Ω.

Szkic

Czas na opracowanie szkicu. W środowisku IDE należy wpisać następujący kod:

```
// Projekt nr 1 — tworzenie fali migających diod LED
❶ void setup()
  {
     pinMode(2, OUTPUT);      // pin sterujący pierwszą diodą LED jest ustawiany
                              // jako wyjściowy
```

```
    pinMode(3, OUTPUT);      // to samo należy zrobić dla diod LED od 2 do 5
    pinMode(4, OUTPUT);
    pinMode(5, OUTPUT);
    pinMode(6, OUTPUT);
}

❷ void loop()
{
    digitalWrite(2, HIGH);  // włącza diodę LED nr 1
    delay(500);             // czeka pół sekundy
    digitalWrite(2, LOW);   // wyłącza diodę LED nr 1
    digitalWrite(3, HIGH);  // ta sama procedura jest powtarzana dla diod LED od 2 do 5
    delay(500);
    digitalWrite(3, LOW);
    digitalWrite(4, HIGH);
    delay(500);
    digitalWrite(4, LOW);
    digitalWrite(5, HIGH);
    delay(500);
    digitalWrite(5, LOW);
    digitalWrite(6, HIGH);
    delay(500);
    digitalWrite(6, LOW);
    digitalWrite(5, HIGH);
    delay(500);
    digitalWrite(5, LOW);
    digitalWrite(4, HIGH);
    delay(500);
    digitalWrite(4, LOW);
    digitalWrite(3, HIGH);
    delay(500);
    digitalWrite(3, LOW);
    // funkcja loop() wróci do swojego pierwszego wiersza i ponownie wykona cały kod
}
```

W ciele funkcji `void setup()` ❶ ustawiono piny cyfrowe wejścia-wyjścia jako piny wyjściowe, ponieważ za ich pośrednictwem system Arduino ma zasilać diody LED (na żądanie programu). To, kiedy poszczególne diody LED mają być włączane, określono za pomocą wywołań funkcji `digitalWrite()` w ciele funkcji `void loop()` ❷.

Schemat

Czas na budowę właściwego obwodu. Układ obwodu można opisać na kilka sposobów. Aby lepiej wyjaśnić schematy pierwszych kilku projektów opisanych w tej książce, zastosuję diagramy ilustrujące fizyczny układ (pierwszy taki diagram pokazano na rysunku 3.13).

Rysunek 3.13. Układ obwodu składającego się na projekt nr 1

Porównując diagram połączeń z funkcjami zaimplementowanymi w szkicu, można łatwo zrozumieć sens obwodu. Na przykład wywołanie funkcji `digital ↪Write(2, HIGH)` powoduje zasilanie pinu cyfrowego nr 2 prądem o napięciu 5 V (stan wysoki) — prąd przepływa przez rezystor ograniczający natężenie, diodę LED (odpowiednio przez anodę i katodę) i wreszcie trafia do gniazda uziemienia (GND) na płytce Arduino. Wywołana chwilę później funkcja `digitalWrite(2, LOW)` wstrzymuje przepływ prądu i — tym samym — powoduje wyłączenie odpowiedniej diody LED.

Uruchamianie szkicu

Należy teraz podłączyć płytkę Arduino do komputera i wysłać szkic. Po krótkiej chwili diody LED powinny migać od lewej do prawej strony i z powrotem (cała procedura będzie powtarzana bez końca). Poczucia osiągniętego sukcesu nie da się porównać z niczym innym — zachęcam do świętowania!

Jeśli jednak diody nie zaczęły migać, należy natychmiast odłączyć przewód USB od płytki Arduino, a następnie sprawdzić, czy szkic został prawidłowo przepisany w środowisku IDE. W razie znalezienia błędu należy go naprawić, po czym ponownie wysłać szkic. Jeśli szkic jest prawidłowy, a mimo to diody LED nie migają, należy sprawdzić wszystkie połączenia na płytce uniwersalnej.

Wiesz już, jak spowodować miganie diod LED za pomocą płytki Arduino, jednak szkic w tej formie jest dość nieefektywny. Gdyby programista chciał na przykład skrócić czas oczekiwania pomiędzy włączaniem i wyłączaniem kolejnych diod LED, musiałby zmodyfikować wszystkie wywołania `delay(500)`. Projekt można zaimplementować w lepszy sposób.

Stosowanie zmiennych

Do przechowywania danych w programach komputerowych służą tzw. *zmienne* (ang. *variables*). Na przykład w szkicu dla projektu nr 1 zastosowano wywołanie funkcji `delay(500)`, aby dioda LED świeciła się przez pół sekundy.

Największą wadą tego szkicu jest brak elastyczności. Aby zmienić czas oczekiwania pomiędzy włączaniem a wyłączaniem diod LED, należałoby ręcznie zmodyfikować wszystkie wywołania funkcji `delay()`. Problem można jednak rozwiązać, tworząc zmienną reprezentującą wartość przekazywaną na wejściu funkcji `delay()`.

W tym celu należy dodać do szkicu dla projektu nr 1 następujący wiersz nad funkcją `void setup()`, ale pod początkowym komentarzem:

```
int d = 250;
```

Powyższy wiersz przypisuje wartość 250 zmiennej nazwanej `d`.

Należy jeszcze zastąpić wszystkie wystąpienia liczby 500 w szkicu odwołaniami do zmiennej `d`. Po ponownym uruchomieniu szkicu system Arduino użyje wartości zmiennej `d` we wszystkich wywołaniach funkcji `delay()`. Po wysłaniu zmienionego szkicu na płytkę Arduino diody LED będą włączane i wyłączane dwukrotnie szybciej, ponieważ czas oczekiwania pomiędzy tymi operacjami skrócono do 250 ms.

Słowo `int` oznacza, że zmienna zawiera liczbę całkowitą (ang. *integer*) z przedziału od −32 768 do 32 767. W największym uproszczeniu liczby całkowite nie obejmują części ułamkowych (miejsc po przecinku). Od tej pory zmiana czasu oczekiwania wymaga tylko modyfikacji deklaracji zmiennej d na początku szkicu. Na przykład przypisanie tej zmiennej liczby 100 jeszcze bardziej przyspieszy włączanie i wyłączanie diod LED:

```
int d = 100;
```

Zachęcam do eksperymentów z tym szkicem — warto sprawdzić różne czasy oczekiwania i zmienione sekwencje wysyłania sygnałów `HIGH` i `LOW`. Takie eksperymenty to mnóstwo zabawy. Nawet po zakończeniu tych testów nie należy rozłączać obwodu — na bazie tego rozwiązania zrealizujemy dalsze projekty w tym rozdziale.

Projekt nr 2: powtarzanie instrukcji za pomocą pętli for

Podczas projektowania szkicu często należy powtarzać, wielokrotnie wykonywać te same funkcje. Oczywiście istnieje możliwość kopiowania i wklejania tych funkcji w ramach szkicu, jednak takie rozwiązanie jest nieefektywne i powoduje nadmierne wykorzystanie pamięci programu w systemie Arduino. Lepszym wyjściem jest zastosowanie pętli for. Zaletą pętli for jest możliwość określenia, ile razy ma być wykonany kod umieszczony wewnątrz tej pętli.

Aby zrozumieć, jak w praktyce działa taka pętla, należy umieścić w nowym szkicu następujący kod:

```
// Projekt nr 2 — powtarzanie instrukcji za pomocą pętli for
int d = 100;

void setup()
{
   pinMode(2, OUTPUT);
   pinMode(3, OUTPUT);
   pinMode(4, OUTPUT);
   pinMode(5, OUTPUT);
   pinMode(6, OUTPUT);
}

void loop()
{
   for ( int a = 2; a < 7 ; a++ )
   {
      digitalWrite(a, HIGH);
      delay(d);
      digitalWrite(a, LOW);
      delay(d);
   }
}
```

Pętla for powtarza kod zawarty pomiędzy nawiasami klamrowymi, dopóki zastosowany warunek jest spełniony. W powyższym kodzie wykorzystano nową zmienną całkowitoliczbową (nazwaną `a`), której początkowa wartość wynosi 2. Za każdym razem, gdy kod pętli jest wykonywany, operacja `a++` zwiększa wartość zmiennej `a` o 1. Kod umieszczony w pętli jest powtarzany tak długo, aż zmienna a ma wartość mniejszą niż 7 (tak zdefiniowano warunek pętli). W momencie, w którym zmienna a osiąga wartość większą lub równą 7, system Arduino przystępuje do wykonywania ewentualnego kodu zdefiniowanego po pętli for.

Liczbę iteracji wykonywanych przez pętlę for można określić także poprzez odliczanie w dół, od większej do mniejszej liczby. Aby zademonstrować tę formę pętli `for`, warto umieścić następujący kod w szkicu dla projektu nr 2 (pod pierwszą pętlą `for`):

```
❶ for ( int a = 5 ; a > 2; a-- )
  {
     digitalWrite(a, HIGH);
     delay(d);
     digitalWrite(a, LOW);
     delay(d);
  }
```

Pętla for w tej formie ❶ przypisuje zmiennej a wartość 5, a następnie odejmuje od tej zmiennej 1 w każdej iteracji pętli za pomocą operacji `a--`. Pętla działa w ten sposób, dopóki wartość zmiennej a jest większa od 2 (`a > 2`); pętla kończy działanie po osiągnięciu przez tę zmienną wartości mniejszej lub równej 2.

Zastosowane zmiany umożliwiły osiągnięcie celów projektu nr 1 za pomocą mniejszej ilości kodu. Warto teraz wysłać kod na platformę Arduino i sprawdzić działanie nowego szkicu!

Zmiana jasności diod LED za pomocą modulacji szerokości impulsu

Zamiast tylko włączania i wyłączania diod za pomocą funkcji `digitalWrite()` istnieje możliwość definiowania poziomu jasności diod LED poprzez określanie czasu upływającego pomiędzy stanami diody LED metodą *modulacji szerokości impulsu* (ang. *pulse-width modulation — PWM*). Metoda PWM może z powodzeniem służyć do tworzenia iluzji różnych poziomów jasności diody LED — wystarczy włączać i wyłączać diodę z częstotliwością około 500 cykli na sekundę. Jasność świecenia diody LED widoczna dla ludzkiego oka zależy od stosunku czasu, w którym pin cyfrowy sterujący tą diodą jest włączony, do czasu, w którym ten sam pin jest wyłączony. Ponieważ ludzkie oko nie dostrzega migotania z częstotliwością przekraczającą 50 cykli na sekundę, użytkownik ma wrażenie, że dioda LED stale świeci.

Im wyższy jest tzw. współczynnik *wypełnienia impulsu* (ang. *duty cycle*), tj. im dłużej odpowiedni pin cyfrowy jest włączony (im rzadziej jest wyłączony), tym jaśniej (przynajmniej w odczuciu użytkownika) świeci dioda LED podłączona do tego pinu wyjściowego.

Różne współczynniki wypełnienia modulacji szerokości impulsu (PWM) pokazano na rysunku 3.14. Wypełnione szare obszary reprezentują czas, w którym dioda LED rzeczywiście jest włączona. Jak widać, czas, w którym dioda LED rzeczywiście świeci, jest tym dłuższy, im wyższy jest współczynnik wypełnienia.

Metoda modulacji szerokości impulsu (PWM) może być stosowana tylko za pośrednictwem pinów cyfrowych nr 3, 5, 6, 9, 10 i 11 na płytce Arduino. Wszystkie te piny oznaczono na płytce Arduino znakiem tyldy (~); patrz rysunek 3.15.

Rysunek 3.14. Różne poziomy współczynnika wypełnienia modulacji szerokości impulsu

Rysunek 3.15. Piny umożliwiające stosowanie modulacji PWM oznaczono symbolem tyldy (~)

Sygnał PWM można wygenerować za pomocą funkcji `analogWrite(x, y)`, gdzie *x* reprezentuje odpowiedni pin cyfrowy, natomiast *y* reprezentuje wartość współczynnika wypełnienia impulsu (z przedziału od 0 do 255, gdzie 0 oznacza 0% współczynnik wypełnienia, a 255 oznacza 100% współczynnik wypełnienia).

Projekt nr 3: przykład zastosowania metody PWM

Spróbujmy teraz zastosować opisaną metodę w obwodzie z projektu nr 2. Wystarczy wpisać poniższy szkic w środowisku IDE i wysłać gotowy kod na platformę Arduino:

```
// Projekt nr 3 — przykład zastosowania metody PWM
int d = 5;
void setup()
{
   pinMode(3, OUTPUT);   // dioda LED jest sterowana przez pin nr 3 (przystosowany
                         // do obsługi modulacji PWM)
}
```

```
void loop()
{
   for ( int a = 0 ; a < 256 ; a++ )
   {
      analogWrite(3, a);
      delay(d);
   }

   for ( int a = 255 ; a >= 0 ; a-- )
   {
      analogWrite(3, a);
      delay(d);
   }
   delay(200);
}
```

Dioda LED podłączona do pinu cyfrowego nr 3 będzie płynnie zwiększała i zmniejszała jasność wraz z rosnącym i spadającym współczynnikiem wypełnienia impulsu. Innymi słowy, dioda LED zostanie włączona, będzie świeciła coraz jaśniej (aż do osiągnięcia maksymalnej jasności), a następnie będzie płynnie gasła (aż do całkowitego wyłączenia). Zachęcam do samodzielnych eksperymentów z tym szkicem i obwodem. Warto na przykład zastosować pięć diod LED rozjaśnianych i ściemnianych jednocześnie lub w określonej sekwencji.

Dodatkowe komponenty elektroniczne

Samo planowanie sterowania urządzeniami za pośrednictwem cyfrowych pinów wyjściowych (bez uwzględniania wymaganego natężenia prądu) zwykle jest dość proste. Podczas właściwej realizacji projektu musimy jednak pamiętać, że każdy z cyfrowych pinów wyjściowych na płytce Arduino Uno może dostarczać prąd o natężeniu 40 mA (wszystkie te piny łącznie dostarczają prąd o natężeniu 200 mA). Okazuje się jednak, że istnieją trzy komponenty sprzętowe ułatwiające zarządzanie natężeniem generowanym przez piny cyfrowe na płytce Arduino — wszystkie te komponenty zostaną omówione w poniższych punktach.

OSTRZEŻENIE *Próba przekroczenia natężenia 40 mA na pojedynczym pinie lub natężenia 200 mA na wszystkich pinach może doprowadzić do uszkodzenia układu scalonego mikrokontrolera.*

Tranzystor

Niemal każdy słyszał o *tranzystorze*, jednak większość nie wie, jak naprawdę działa to urządzenie. Aby uniknąć długich prezentacji, postaram się możliwie najprościej wyjaśnić działanie tranzystora. Tranzystor umożliwia włączanie lub wyłączanie przepływu prądu o natężeniu dużo większym niż to obsługiwane przez

platformę Arduino Uno. Okazuje się jednak, że działaniem tranzystora można bezpiecznie sterować za pośrednictwem cyfrowych pinów wyjściowych na płytce Arduino. Przykład popularnego tranzystora, BC548, pokazano na rysunku 3.16.

Rysunek 3.16. Typowy tranzystor: BC548

Podobnie jak w przypadku diody LED, każda nóżka tranzystora pełni określoną funkcję, zatem tranzystor należy podłączyć do obwodu w odpowiedniej orientacji. Jeśli patrzymy na płaską stronę tranzystora (tak jak po lewej stronie rysunku 3.16), piny tranzystora BC548 nazywa się (od lewej do prawej strony) *kolektorem* (*C*), *bazą* (*B*) i *emiterem* (*E*). (Warto pamiętać, że opisany układ pinów dotyczy tego konkretnego tranzystora; inne tranzystory mogą wymagać odmiennego sposobu łączenia z obwodem). Jeśli prąd o niewielkim natężeniu trafia do bazy tranzystora (na przykład za pośrednictwem cyfrowego pinu wejścia-wyjścia na płytce Arduino), a prąd o większym natężeniu przechodzi przez kolektor, oba natężenia są łączone i przekazywane dalej za pośrednictwem emitera. W momencie wyłączenia natężenia sterującego przekazywanego do bazy tranzystor w ogóle nie przekazuje prądu do emitera.

Tranzystor BC548 może służyć do przełączania prądu o maksymalnym natężeniu 100 mA i maksymalnym napięciu 30 V, czyli dużo większych parametrach niż parametry prądu przekazywanego przez cyfrowe piny wyjściowe na płytce Arduino. W projektach opisanych w dalszej części tej książki będą stosowane także inne tranzystory — każdorazowo będę dokładnie wyjaśniał działanie tych komponentów.

UWAGA *Zawsze należy mieć na uwadze układ pinów tranzystora, ponieważ każdy typ tranzystora może wymagać łączenia w nieco innej orientacji.*

Dioda prostownicza

Dioda prostownicza jest bardzo prostym, a mimo to wyjątkowo przydatnym komponentem. Prąd może przepływać przez diodę prostowniczą tylko w jednym kierunku. Swoim wyglądem dioda prostownicza pod wieloma względami przypomina rezystor (patrz rysunek 3.17).

W projektach opisanych w tej książce będą stosowane diody prostownicze typu 1N4004. Prąd trafia do diody prostowniczej za pośrednictwem anody i opuszcza ją za pośrednictwem katody (odpowiednia strona diody jest oznaczona pierścieniem na obudowie). Diody prostownicze chronią elementy obwodu przed

Rysunek 3.17. Dioda prostownicza typu 1N4004

przepływem natężenia w przeciwnym kierunku. Diody prostownicze mają jednak pewną wadę — powodują spadek napięcia o około 0,7 V. Dioda 1N4004 może obsługiwać prąd o natężeniu 1 A i napięciu 400 V, czyli dużo wyższych parametrach niż te stosowane w opisywanych tutaj obwodach. Zaletą tej diody jest jej popularność i niewysoka cena.

Przekaźnik

Przekaźniki są stosowane w tym samym celu co tranzystory — umożliwiają kontrolę dużo wyższego natężenia i napięcia. Zaletą przekaźnika elektrycznego jest izolacja od obwodu sterującego, dzięki czemu na przykład płytka Arduino może przełączać prąd o bardzo wysokim natężeniu i napięciu. W niektórych przypadkach taka izolacja jest niezbędna do ochrony obwodów przed dużymi natężeniami i napięciami, które mogą uszkodzić płytkę Arduino. Wewnątrz przekaźnika znajdują się dwa interesujące elementy: mechaniczny przełącznik oraz cewka zasilana niskim napięciem (patrz rysunek 3.18).

Rysunek 3.18. Budowa typowego przekaźnika

Kiedy prąd przepływa przez cewkę, komponent staje się elektromagnesem przyciągającym metalowy element (który pełni funkcję swoistej dźwigni przełącznika). Włączony elektromagnes (pod wpływem prądu przepływającego przez cewkę) przyciąga metalowy element, natomiast po wyłączeniu (odcięciu prądu od cewki) element wraca do pozycji wyjściowej. Ruch tej dźwigni powoduje charakterystyczny dźwięk kliknięcia (ten sam, który wydawały kierunkowskazy w starszych samochodach).

Obwody zasilane wyższym napięciem

Skoro rozumiesz już ogólne działanie tranzystora, diody prostowniczej i przekaźnika, warto podjąć próbę łącznego wykorzystania tych komponentów do sterowania wyższymi natężeniami i napięciami. Łączenie wymienionych komponentów jest bardzo proste (patrz rysunek 3.19).

Rysunek 3.19. Obwód z przekaźnikiem

Ten prosty obwód obejmuje przekaźnik z cewką zasilaną napięciem 12 V. Jednym z możliwych zastosowań tego obwodu jest sterowanie lampą lub wentylatorem systemu chłodzenia podłączonym do styków przekaźnika. Pin cyfrowy nr 10 na płytce Arduino połączono z bazą tranzystora za pośrednictwem rezystora 1 kΩ. Tranzystor steruje prądem przepływającym przez cewkę przekaźnika (włącza lub wyłącza przepływ prądu). Warto pamiętać, że nóżki tranzystora reprezentują odpowiednio końcówki *C*, *B* i *E*, pod warunkiem że patrzymy na płaską stronę tranzystora. Obiekt po lewej stronie płytki uniwersalnej ❶ reprezentuje zasilacz 12 V podłączony do cewki przekaźnika. Uziemienie (minus) ❷ zasilacza 12 V jest połączone z pinem emitera tranzystora oraz pinem GND na płytce GND. I wreszcie dioda prostownicza 1N4004 jest połączona z cewką przekaźnika ❸; katodę diody połączono z dodatnią stroną zasilacza. Piny prowadzące do styków przekaźnika można znaleźć w karcie produktu — aby wszystko było połączone prawidłowo, wystarczy postępować według danych zawartych w karcie.

Dioda prostownicza ma na celu ochronę obwodu. W chwili odcięcia zasilania cewki przekaźnika przez moment w obwodzie występuje zjawisko tzw. prądu błądzącego — następuje skok napięcia, które należy gdzieś skierować. Dioda powoduje, że prąd błądzący wielokrotnie przechodzi przez cewkę aż do ostatecznego

zaniknięcia (zamiany na niewielką ilość energii cieplnej). Opisane rozwiązanie zabezpiecza przed uszkodzeniem zarówno tranzystor, jak i płytkę Arduino.

OSTRZEŻENIE *Jeśli przekaźnik ma służyć do przełączania prądu sieciowego (110 – 250 V), należy poprosić o pomoc elektryka dysponującego odpowiednimi uprawnieniami. Nawet najdrobniejsza pomyłka może się fatalnie skończyć dla niedoświadczonego elektronika.*

Co dalej?

Rozdział 3. zbliża się do końca. Mam nadzieję, że budowa przykładowych projektów i samodzielne eksperymenty z diodami LED sprawiły Ci sporo radości. W tym rozdziale koncentrowaliśmy się na różnych sposobach sterowania migającymi diodami LED za pomocą systemu Arduino, modyfikowaliśmy istniejące systemy oraz poznaliśmy funkcje i pętle, czyli konstrukcje programistyczne umożliwiające bardziej efektywne sterowanie komponentami podłączonymi do płytki Arduino. Wiedza zdobyta podczas lektury tego rozdziału będzie bardzo pomocna w dalszych rozdziałach.

Realizacja projektów opisanych w rozdziale 4. będzie wyjątkowo interesującym doświadczeniem. Zbudujemy tak praktyczne systemy jak uliczna sygnalizacja świetlna, termometr, tester baterii — jeśli więc jesteś gotowy wspiąć się na kolejny poziom, zachęcam do przewrócenia strony!

4

Elementy składowe obwodów

W tym rozdziale:

- nauczysz się czytać schematy obwodów, czyli rozumieć język opisywania obwodów elektronicznych;
- poznasz kondensator;
- opanujesz metody stosowania pinów wejściowych;
- zaczniesz stosować działania arytmetyczne i sprawdzać wartość zmiennych;
- zaimplementujesz podejmowanie decyzji w formie wyrażeń `if-then-else`;
- zrozumiesz różnice dzielące sygnał analogowy od sygnału cyfrowego;
- nauczysz się mierzyć analogowe źródła napięcia na różnych poziomach precyzji;

- poznasz rezystory nastawne, brzęczyki piezoelektryczne oraz czujniki temperatur;
- uporządkujesz zdobytą wiedzę, budując system sygnalizacji świetlnej, tester baterii i termometr.

Informacje zawarte w tym rozdziale powinny znacznie ułatwić zrozumienie potencjału platformy Arduino. W tym rozdziale nadal będę wprowadzał podstawy elektroniki, w tym nowe komponenty, sposób czytania schematów obwodów (swoistych map drogowych obwodów elektronicznych) oraz rodzaje mierzalnych sygnałów elektrycznych. W dalszej części tego rozdziału omówię dodatkowe funkcje systemu Arduino, w tym techniki przechowywania wartości, wykonywania działań matematycznych i podejmowania decyzji. I wreszcie omówię dodatkowe komponenty, a następnie pokażę, jak ich używać w praktyce (na przykładzie projektów realizujących pewne przydatne funkcje).

Stosowanie schematów obwodów

W rozdziale 3. omawiałem budowę obwodów elektronicznych, stosując schematy fizycznego układu komponentów zamontowanych na płytce uniwersalnej. Mimo że takie schematy fizyczne mogą się wydawać najprostszym sposobem opisywania obwodów, jednak po dodaniu większej liczby komponentów schematy w tej formie stałyby się zupełnie nieczytelne. Ponieważ także nasze obwody będą coraz bardziej skomplikowane, w tym rozdziale zacznę się posługiwać schematami obwodów (ang. *schematic diagrams*, *circuit diagrams*); przykład takiego schematu pokazano na rysunku 4.1.

Rysunek 4.1. Przykładowy schemat obwodu

Schematy obwodów są w istocie mapami drogowymi, które opisują ścieżkę przepływu prądu elektrycznego przez poszczególne komponenty. Zamiast zdjęć komponentów i przewodów schemat obwodu zawiera odpowiednie symbole i linie.

Identyfikacja komponentów

Po opanowaniu znaczenia poszczególnych symboli odczytywanie schematów obwodów jest dość proste. W pierwszej kolejności wprowadzę symbole reprezentujące komponenty, które były już stosowane w dotychczasowych projektach.

Płytka Arduino

Na rysunku 4.2 pokazano symbol samej płytki Arduino. Jak widać, wszystkie połączenia tej płytki są oznaczone odpowiednimi etykietami.

Rysunek 4.2. Symbol płytki Arduino Uno

Rezystor

Symbol rezystora pokazano na rysunku 4.3.

Oprócz samego symbolu rezystora dobrą praktyką jest umieszczanie na schemacie wartości rezystancji i numeru komponentu (w tym przypadku 220 Ω i R1). Dzięki temu interpretacja schematu obwodu będzie dużo łatwiejsza dla każdego, kto będzie się musiał z nim zapoznać (także dla jego twórcy). Na niektórych schematach obwodów omy są zastępowane przez wielką literę *R*, na przykład 220 R.

Rysunek 4.3. Symbol rezystora

Dioda prostownicza

Symbol diody prostowniczej pokazano na rysunku 4.4.

Jak napisałem w rozdziale 3., diody prostownicze są komponentami spolaryzowanymi, a prąd przepływa od anody do katody. Na symbolu pokazanym na rysunku 4.4 anoda znajduje się po lewej stronie, natomiast katoda po prawej. Najprostszym sposobem, by to za-

Rysunek 4.4. Symbol diody prostowniczej

pamiętać, jest przyjęcie zasady, zgodnie z którą prąd płynie tylko w kierunku wierzchołka trójkąta. Prąd nie może płynąć w przeciwnym kierunku, ponieważ jest „blokowany” przez pionową linię.

Dioda LED

Symbol diody LED pokazano na rysunku 4.5.

Wszystkie diody są oznaczane tym samym symbolem — trójkątem i pionową linią. Okazuje się jednak, że symbole diod LED dodatkowo zawierają dwie równoległe strzałki sygnalizujące emitowane światło.

Rysunek 4.5. Symbol diody LED

Tranzystor

Symbol tranzystora pokazano na rysunku 4.6. Symbol będzie używany do reprezentowania tranzystora BC548.

Pionowa linia na górze tego symbolu (oznaczona literą *C*) reprezentuje kolektor, pozioma linia z lewej strony (oznaczona literą *B*) reprezentuje bazę, a pionowa linia na dole (oznaczona literą *E*) reprezentuje emiter. Strzałka w środkowej części symbolu (skierowana w prawą stronę w dół) wskazuje, że jest to tranzystor typu NPN, ponieważ tylko w tranzystorach *NPN* prąd może przepływać od kolektora do emitera. (W tranzystorach *PNP* prąd może przepływać od emitera do kolektora).

Rysunek 4.6. Symbol tranzystora

Do numerowania tranzystorów należy używać litery *Q* (tak jak do numerowania rezystorów służy litera *R*).

Przekaźnik

Symbol przekaźnika pokazano na rysunku 4.7.

Symbole przekaźników mają różne formy i mogą obejmować odmienne zestawy styków. Pewne elementy są jednak wspólne dla wszystkich symboli przekaźników. Pierwszym takim elementem jest cewka (reprezentowana przez pionową krzywą po lewej stronie). Drugim elementem wspólnym są styki przekaźnika. Styk *COM* (wspólny, od ang. *common*) zwykle jest używany jako wejście, natomiast styki oznaczone etykietami *NO* (standardowo otwarty, od ang. *normally open*) i *NC* (standardowo zamknięty, od ang. *normally closed*) najczęściej są używane jako wyjścia.

Rysunek 4.7. Symbol przekaźnika

Symbol zawsze reprezentuje stan wyłączonego przekaźnika (kiedy *cewka nie jest zasilana*), czyli z połączonymi pinami COM i NC. Po włączeniu zasilania cewki zostaną połączone piny COM i NO (symbol reprezentujący ten stan wyglądałby nieco inaczej).

Połączenia na schemacie obwodu

Połączenia (przewody) na schemacie mogą się łączyć lub przecinać — w obu przypadkach należy stosować nieco inne oznaczenia (patrz poniższe przykłady).

Przecinające się, ale niepołączone przewody

Jeśli dwa przewody przecinają się, ale nie są połączone, punkt przecięcia może być reprezentowany na jeden z dwóch sposobów (patrz rysunek 4.8). Żaden z symboli nie jest lepszy od drugiego — wybór zależy wyłącznie od preferencji autora schematu.

Rysunek 4.8. Niepołączone, przecinające się przewody

Połączone przewody

Jeśli przewody zostały fizycznie połączone, *miejsce łączenia* należy oznaczyć wyraźną kropką (patrz rysunek 4.9).

Rysunek 4.9. Dwa połączone przewody

Przewody połączone z uziemieniem

Jeśli przewód jest uziemiony (połączony z uziemieniem, zwykle oznaczanym jako GND), standardową reprezentacją tego stanu jest symbol pokazany na rysunku 4.10.

Rysunek 4.10. Symbol uziemienia (GND)

Symbol uziemienia na końcu linii na schemacie obwodu informuje o fizycznym podłączeniu odpowiedniego przewodu do pinu GND na płytce Arduino.

Analiza przykładowego schematu

Skoro wyjaśniłem już znaczenie symboli reprezentujących różne komponenty oraz sposoby graficznego reprezentowania połączeń, czas przystąpić do szczegółowej analizy schematu obwodu narysowanego z myślą o projekcie nr 1. Celem projektu było kolejne włączanie i wyłączanie pięciu diod LED.

Warto porównać schemat obwodu pokazany na rysunku 4.11 z rysunkiem 3.13 — trudno nie zgodzić się z tym, że schemat obwodu jest dużo prostszym sposobem opisywania projektu niż metoda stosowana w rozdziale 3.

Od tej pory będę opisywał obwody wyłącznie za pomocą schematów. Przy okazji wprowadzania nowych komponentów będę też prezentował i opisywał odpowiednie symbole.

UWAGA *Czytelników zainteresowanych tworzeniem własnych schematów zachęcam do zapoznania się z aplikacją Fritzing, która jest dostępna za darmo na stronie http://www.fritzing.org/.*

Rysunek 4.11. Schemat obwodu dla projektu nr 1

Kondensator

Kondensator to urządzenie utrzymujące ładunek elektryczny. Kondensator składa się z dwóch metalowych płytek i warstwy izolującej, która umożliwia zgromadzenie ładunku elektrycznego pomiędzy wspomnianymi płytkami (tzw. okładkami). W momencie odcięcia zasilania ładunek elektryczny jest zachowywany i może zostać odebrany z kondensatora (w procesie tzw. *rozładowania*) — wystarczy, że zostanie zapewniona nowa ścieżka przepływu prądu w wyniku powstania napięcia pomiędzy okładkami kondensatora.

Mierzenie pojemności kondensatora

Ładunek elektryczny, który może być utrzymywany przez kondensator, wyraża się w *faradach*. Warto przy tym podkreślić, że jeden farad to bardzo duży ładunek. Oznacza to, że pojemność zdecydowanej większości kondensatorów jest mierzona w pikofaradach i mikrofaradach.

Jeden *pikofarad* (pF) to 0,000000000001 farada, a jeden *mikrofarad* (μF) to 0,000001 farada. Kondensatory są wytwarzane z myślą o określonych napięciach maksymalnych. W tej książce będą używane tylko kondensatory niskonapięciowe, o maksymalnym napięciu zasilania nie większym niż 10 V. Warto jednak pamiętać, że stosowanie kondensatorów o wyższym deklarowanym napięciu maksymalnym

w obwodach zasilanych niższym napięciem nie stanowi problemu. Typowe deklarowane napięcia maksymalne wynoszą 10, 16, 25 i 50 V.

Odczytywanie wartości kondensatorów

Odczytanie wartości kondensatora ceramicznego wymaga pewnej praktyki, ponieważ parametry kondensatora są zakodowane. Dwie pierwsze cyfry reprezentują wartość wyrażoną w pikofaradach, natomiast trzecia cyfra reprezentuje mnożnik (liczbę dziesiątek, przez które należy pomnożyć pierwszą liczbę). Na przykład kondensator pokazany na rysunku 4.12 oznaczono liczbą *104*. Oznacza to, że pojemność kondensatora to 10 i cztery zera, czyli 100 000 pikofaradów (pF) lub 100 nanofaradów (nF), lub 0,1 mikrofarada (μF).

Rysunek 4.12. Kondensator ceramiczny o pojemności 0,1 μF

UWAGA *Konwersje jednostek miar mogą rodzić pewne nieporozumienia, zatem warto wydrukować doskonałą tabelę dostępną na stronie http://www.justradios.com/uFnFpF.html.*

Rodzaje kondensatorów

W projektach opisywanych w tej książce będą stosowane dwa typy kondensatorów: ceramiczne i elektrolityczne.

Kondensatory ceramiczne

Kondensatory ceramiczne, podobne do tego pokazanego na rysunku 4.12, są bardzo małe i mogą utrzymywać niewielkie ładunki elektryczne. Kondensatory tego typu nie są spolaryzowane, zatem prąd może przez nie przechodzić w obu kierunkach. Symbol niespolaryzowanego kondensatora stosowany na schematach obwodów pokazano na rysunku 4.13.

Rysunek 4.13. Symbol niespolaryzowanego kondensatora (pojemność kondensatora należy zapisać w prawym górnym rogu symbolu)

Kondensatory ceramiczne doskonale sprawdzają się w obwodach o wysokiej częstotliwości, ponieważ (dzięki niewielkiej pojemności) mogą być błyskawicznie ładowane i rozładowywane.

Kondensatory elektrolityczne

Kondensatory elektrolityczne, podobne do tego pokazanego na rysunku 4.14, mają większe wymiary fizyczne od kondensatorów ceramicznych, oferują większe pojemności i są spolaryzowane. Na obudowie kondensatorów tego typu można znaleźć oznaczenie strony dodatniej (+) lub ujemnej (–). Na rysunku 4.14 widać jasny pasek i mały symbol minusa (–) wskazujący ujemną końcówkę kondensatora. Podobnie jak rezystory, kondensatory cechują się (oprócz podstawowych parametrów) poziomem tolerancji. Kondensator pokazany na rysunku 4.14 ma 20-procentową tolerancję i pojemność 100 μF.

Rysunek 4.14. Kondensator elektrolityczny

Symbol kondensatora elektrolitycznego (patrz rysunek 4.15) zawiera symbol plusa (+) wskazujący polaryzację kondensatora.

Rysunek 4.15. Symbol kondensatora spolaryzowanego

Kondensatory elektrolityczne często są używane do utrzymywania większych ładunków elektrycznych do stabilizacji napięcia dostarczanego przez zasilacz. Kondensatory tego typu, niczym niewielkie, tymczasowe baterie, mogą z powodzeniem służyć do stabilizacji napięcia w bezpośrednim sąsiedztwie obwodów lub części narażonych na chwilowe przerwy w dostarczaniu prądu przez zasilacz. Oznacza to, że kondensatory ograniczają skutki niechcianych spadków napięcia i zakłóceń w obwodach. Parametry kondensatora elektrolitycznego są drukowane bezpośrednio na tych komponentach i nie wymagają dekodowania ani specjalnej interpretacji.

Ponieważ wyjaśniłem już, jak generować podstawowe formy sygnałów wyjściowych przy użyciu diod LED podłączonych do płytki Arduino, czas opanować metody obsługi cyfrowych pinów wejściowych na tej płytce, aby stworzyć system podejmujący decyzje w zależności od otrzymywanych danych wejściowych.

Wejście cyfrowe

W rozdziale 3. piny cyfrowe wejścia-wyjścia były używane w roli wyjścia umożliwiającego włączanie i wyłączanie diod LED. Te same piny mogą służyć do otrzymywania danych wejściowych od użytkowników, na przykład wykrywania naciśnięcia przycisku przez użytkownika.

MIERZENIE DRGAŃ STYKÓW PRZEŁĄCZNIKA ZA POMOCĄ OSCYLOSKOPU CYFROWEGO Z PAMIĘCIĄ

W przyciskach występuje zjawisko nazywane drganiem styków (ang. *switch bounce*, *bouncing*) i polegające na wielokrotnym włączaniu i wyłączaniu przycisku przez pewien czas po naciśnięciu przez użytkownika. Powodem tego zjawiska jest niewielki rozmiar metalowych styków wewnątrz przycisków — styki drgają bezpośrednio po naciśnięciu lub zwolnieniu przycisku, powodując przez krótki czas wielokrotne włączanie i wyłączanie przycisku.

Drganie styków można zademonstrować na *oscyloskopie cyfrowym z pamięcią* (ang. *digital storage oscilloscope — DSO*), czyli urządzeniu wyświetlającym zmiany napięcia w wybranym okresie. Przykład zjawiska na wyświetlaczu oscyloskopu DSO pokazano na rysunku 4.18.

Rysunek 4.18. Mierzenie drgań styków przycisku

Górna część wyświetlacza widocznego na rysunku 4.18 reprezentuje wynik wielokrotnego naciśnięcia przycisku. W miejscu, w którym pozioma linia reprezentująca napięcie (wskazywana przez strzałkę) znajduje się na wyższym poziomie (5 V), przycisk jest wciśnięty i umożliwia przepływ prądu. Pod słowem Stop widać wycinek czasu bezpośrednio po zwolnieniu przycisku — wycinek jest wyznaczany przez dwie pionowe linie. Poziom napięcia w tym krótkim okresie jest powiększony w dolnej części ekranu. W punkcie A przycisk jest zwalniany przez użytkownika, stąd spadek napięcia do poziomu 0 V (reprezentowanego przez poziomą linię). Okazuje się jednak, że wskutek fizycznych wibracji styku przycisk ponownie wraca do pozycji załączonej, co powoduje ponowne osiągnięcie napięcia 5 V aż do punktu B. Dalsze drgania styku powodują chwilowe rozłączenie i ponowne załączenie przycisku aż do punktu C, gdzie ostatecznie napięcie spada do poziomu 0 V (stanu niskiego). W efekcie zamiast jednego zdarzenia naciśnięcia przycisku w tym przypadku system Arduino otrzyma sygnały o trzech naciśnięciach w bardzo krótkim czasie.

Tak jak cyfrowe piny wyjściowe, także piny wejściowe mogą się znajdować w dwóch stanach: wysokim i niskim. Najprostszą formą cyfrowego wejścia jest *przycisk* (przykłady takich komponentów wejściowych pokazano na rysunku 4.16). Przyciski można umieszczać bezpośrednio na płytce uniwersalnej. Wciśnięty przycisk umożliwia przepływ prądu (wyłączony przycisk odcina napięcie), a cyfrowe piny wejściowe mogą służyć do wykrywania obecności napięcia i — tym samym — określania, czy przycisk jest wciśnięty.

Rysunek 4.16. Proste przyciski zainstalowane na płytce uniwersalnej

Warto zwrócić uwagę na przycisk widoczny na dole rysunku — komponent umieszczono na płytce uniwersalnej w taki sposób, że łączy wiersze 23. i 25. Naciśnięcie tego przycisku spowoduje połączenie obu tych wierszy. Symbol stosowany na schematach obwodów dla tego konkretnego przycisku pokazano na rysunku 4.17. Symbol reprezentuje obie strony przycisku, a sam przycisk jest oznaczany przedrostkiem S i numerem komponentu. Po naciśnięciu przycisku obie strony są łączone, zatem przycisk przepuszcza napięcie i natężenie.

Rysunek 4.17. Symbol przycisku

Projekt nr 4: przykład użycia cyfrowego wejścia

Celem tego projektu jest zastosowanie przycisku, którego włączenie spowoduje zapalenie diody LED na pół sekundy.

Algorytm

Oto algorytm dla tego projektu:

1. Sprawdź, czy przycisk został wciśnięty.
2. Jeśli przycisk jest wciśnięty, włącz diodę LED na pół sekundy, a następnie wyłącz tę diodę.

3. Jeśli przycisk nie został wciśnięty, nie wykonuj żadnych działań.
4. Powtarzaj ten proces w nieskończoność.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- jeden przycisk;
- jedna dioda LED;
- jeden rezystor 560 Ω;
- jeden rezystor 10 kΩ;
- jeden kondensator 100 nF;
- różne przewody;
- płytka uniwersalna;
- płytka Arduino i przewód USB.

Schemat obwodu

Prace nad projektem rozpoczniemy od przygotowania obwodu na płytce uniwersalnej na podstawie schematu obwodu pokazanego na rysunku 4.19. Warto zwrócić uwagę na sposób połączenia rezystora 10 kΩ pomiędzy uziemieniem (pinem GND) a pinem cyfrowym nr 7. Tak podłączony opornik określa się mianem *rezystora ściągającego* (ang. *pull-down resistor*), ponieważ obniża napięcie na pinie cyfrowym niemal do zera. Co więcej, zastosowanie dodatkowego kondensatora 100 nF równolegle do rezystora 10 kΩ pozwala utworzyć prosty obwód filtrujący, który *zabezpiecza system przed zjawiskiem drgających styków przycisku*. W momencie naciśnięcia przycisku pin cyfrowy natychmiast otrzymuje sygnał o stanie wysokim. Jeśli jednak przycisk zostanie zwolniony, pin cyfrowy numer 7 jest łączony z uziemieniem (GND) za pośrednictwem rezystora 10 kΩ, a kondensator 100 nF zapewnia niewielkie opóźnienie. Opisany obwód rozwiązuje problem następujących po sobie niechcianych impulsów poprzez spowolnienie spadku napięcia do poziomu 0 V — w ten sposób można wyeliminować większość fałszywych sygnałów związanych ze zmieniającym się napięciem i błędnymi wskazaniami stanu przycisku.

Ponieważ to pierwszy projekt, w którym obwód będzie budowany na podstawie schematu obwodu, warto postępować według poniższych precyzyjnych instrukcji — dzięki temu zrozumienie sposobu łączenia poszczególnych komponentów powinno być dużo łatwiejsze:

1. Należy umieścić przycisk na płytce uniwersalnej (patrz rysunek 4.20).
2. Należy obrócić płytkę uniwersalną o 90 stopni w kierunku odwrotnym do kierunku ruchu wskazówek zegara, a następnie umieścić na tej płytce rezystor 10 kΩ, krótki przewód i kondensator (patrz rysunek 4.21).

Rysunek 4.19. Schemat dla projektu nr 4

Rysunek 4.20. Przycisk zainstalowany na płytce uniwersalnej

Rysunek 4.21. Rezystor 10 kΩ, kondensator i przycisk

3. Należy połączyć przewodem pin 5 V na płytce Arduino ze skrajnie lewą pionową kolumną na płytce uniwersalnej, a następnie połączyć drugim przewodem pin GND na płytce Arduino z drugą pionową kolumną od lewej strony (na prawo od kolumny połączonej z pinem 5 V; patrz rysunek 4.22).

Rysunek 4.22. Przewody połączone z pinami 5 V (czerwony) i GND (czarny)

4. Należy połączyć pin cyfrowy nr 7 na płytce Arduino z wtykiem na płytce uniwersalnej na prawo od przycisku (patrz rysunek 4.23).

Rysunek 4.23. Połączenie przycisku z cyfrowym pinem wejściowym

5. Diodę LED należy zainstalować na płytce uniwersalnej w taki sposób, aby krótsza nóżka (katoda) była połączona z uziemioną kolumną (połączoną z pinem GND) i aby dłuższa nóżka (anoda) była połączona z kolumną po prawej stronie. Na prawo od diody LED należy umieścić rezystor 560 Ω (patrz rysunek 4.24).

Rysunek 4.24. Sposób instalacji diody LED i rezystora 560 Ω

6. Należy jeszcze połączyć przewodem wtyk na prawo od rezystora 560 Ω z pinem cyfrowym nr 12 na płytce Arduino (patrz rysunek 4.25).

Rysunek 4.25. Połączenie diody LED z płytką Arduino

Przed przystąpieniem do dalszych kroków warto dokładnie sprawdzić schemat obwodu i raz jeszcze przeanalizować połączenia wszystkich komponentów. W szczególności należy sprawdzić zgodność fizycznych połączeń ze schematem obwodu.

Szkic

Na potrzeby tego projektu należy przepisać i wysłać na płytkę Arduino szkic z listingu 4.1.

Listing 4.1. Obsługa cyfrowego wejścia

```
// Projekt nr 4 — przykład użycia cyfrowego wejścia
❶ #define LED 12
#define BUTTON 7

void setup()
{
❷   pinMode(LED, OUTPUT);    // pin wyjściowy dla diody LED
    pinMode(BUTTON, INPUT); // pin wejściowy dla przycisku
}

void loop()
{
   if ( digitalRead(BUTTON) == HIGH )
   {
      digitalWrite(LED, HIGH); // włącza diodę LED
      delay(500);              // czeka pół sekundy
      digitalWrite(LED, LOW);  // wyłącza diodę LED
}
}
```

Po wysłaniu szkicu na płytkę i naciśnięciu przycisku dioda LED powinna świecić przez pół sekundy.

Modyfikowanie szkicu

Skoro mamy już na koncie pierwszy sukces, warto podjąć próbę modyfikacji tego szkicu polegającej na przykład na wydłużeniu czasu świecenia diody LED lub spróbować zastosować przycisk we wcześniejszym projekcie nr 3. (Nie należy jednak rozłączać tego obwodu, ponieważ będzie potrzebny w następnym przykładzie).

Wyjaśnienie szkicu

Przeanalizujmy teraz nowe elementy, które zastosowano w szkicu dla projektu nr 4, w szczególności konstrukcję `#define`, cyfrowe piny wejściowe oraz wyrażenie `if-then`.

Tworzenie stałych za pomocą wyrażenia #define

Przed zdefiniowaniem funkcji `void setup()` zastosowano wyrażenia `#define` ❶, aby utworzyć tzw. stałe — w czasie kompilacji szkicu środowisko IDE zastąpi wszystkie wystąpienia tak zdefiniowanego słowa odpowiednią liczbą. Na przykład po znalezieniu słowa LED w wierszu ❷ środowisko zastąpi to słowo liczbą 12.

W powyższym szkicu wyrażenie `#define` zostało użyte do oznaczenia pinów cyfrowych odpowiedzialnych za wysyłanie sygnałów do diody LED i odbieranie sygnałów od przycisku. Warto zwrócić uwagę na brak średnika po wartości zdefiniowanej w wyrażeniu `#define`. Reprezentowanie numerów pinów (i innych stałych wartości, na przykład czasu oczekiwania) w ten sposób jest szczególnie korzystne w sytuacji, gdy te wartości są wielokrotnie stosowane w szkicu — dzięki temu ewentualna zmiana wymaga modyfikacji kodu tylko w jednym miejscu. W tym przypadku stała LED została użyta w szkicu trzykrotnie; aby zmienić numer pinu, wystarczy zmodyfikować tylko odpowiednią definicję w ramach wyrażenia `#define`.

Odczytywanie stanu pinów cyfrowych

Aby odczytać stan przycisku, należy najpierw zdefiniować (w funkcji `void setup()`) cyfrowy pin wejścia-wyjścia jako pin wejściowy:

```
pinMode(BUTTON, INPUT); // pin wejściowy dla przycisku
```

Aby następnie określić, czy przez przycisk przepływa prąd i trafia do cyfrowego pinu wejściowego (czy przycisk jest wciśnięty), wystarczy użyć funkcji `digitalRead(pin)`, gdzie *`pin`* to numer sprawdzanego pinu cyfrowego. Funkcja zwraca albo wartość `HIGH` (jeśli napięcie na pinie jest zbliżone do 5 V), albo wartość `LOW` (jeśli napięcie jest bliskie 0 V).

Podejmowanie decyzji za pomocą wyrażenia if

Wyrażenie `if` umożliwia podejmowanie decyzji w ramach szkicu, tak aby system Arduino wykonywał inny kod zależnie od sytuacji. Na przykład w szkicu dla projektu nr 4 zastosowano konstrukcję, którą ponownie pokazano na listingu 4.2.

Listing 4.2. Prosty przykład wyrażenia if-then

```
// Listing 4.2
if (digitalRead(BUTTON) == HIGH)
{
      digitalWrite(LED, HIGH);    // włącza diodę LED
      delay(500);                 // czeka pół sekundy
      digitalWrite(LED, LOW);     // wyłącza diodę LED
}
```

Pierwszy wiersz tego kodu zawiera wyrażenie `if` testujące pewien warunek. Jeśli ten warunek jest spełniony (jeśli napięcie ma wartość `HIGH`), przycisk został wciśnięty, zatem należy wykonać kod otoczony nawiasami klamrowymi.

Aby określić, czy przycisk został wciśnięty (czy wywołanie funkcji `digital ↪Read(BUTTON)` zwraca wartość `HIGH`), zastosowano operator porównania w formie podwójnego znaku równości (`==`). Gdyby w tym szkicu zastąpiono operator `==` operatorem różności (`!=`), naciśnięcie przycisku nie spowodowałoby włączenia diody LED. Każdy może to łatwo sprawdzić.

UWAGA *Typowym błędem jest stosowanie pojedynczego znaku równości (=), który oznacza przypisanie wartości z prawej strony do elementu na lewo od tego znaku, zamiast podwójnego znaku równości (==), który oznacza sprawdzenie ewentualnej równości obu stron. W takim przypadku nie zawsze jest generowany komunikat o błędzie, ale tak zapisane wyrażenie z pewnością nie spełni swojej roli!*

Podejmowanie dodatkowych decyzji za pomocą wyrażeń if-then-else

Wyrażenie `if` można uzupełnić o dodatkowe działania, stosując dodatkową klauzulę `else`. Na przykład listing 4.1 można zmodyfikować, dodając element (patrz listing 4.3), tak aby dioda LED włączała się w momencie naciśnięcia przycisku i była wyłączana w momencie zwolnienia przycisku. Wyrażenie `else` powoduje, że system Arduino wykonuje inną sekcję kodu, jeśli warunek zawarty w wyrażeniu `if` nie jest spełniony.

Listing 4.3. Kod uzupełniony o element else

```
// Listing 4.3
#define LED 12
#define BUTTON 7

void setup()
{
   pinMode(LED, OUTPUT); // pin wyjściowy dla diody LED
   pinMode(BUTTON, INPUT); // pin wejściowy dla przycisku
}

void loop()
{
   if ( digitalRead(BUTTON) == HIGH )
   {
      digitalWrite(LED, HIGH);
   }
   else
   {
      digitalWrite(LED, LOW);
   }
}
```

Zmienne logiczne

W pewnych sytuacjach należy reprezentować w programie coś, co może się znajdować tylko w dwóch stanach — na przykład coś, co może być włączone lub wyłączone bądź ciepłe lub zimne. Zmienna logiczna (ang. *boolean variable*) to legendarny bit komputerowy, który może mieć albo wartość 0 (fałsz; ang. *false*), albo wartość 1 (prawda; ang. *true*). Właśnie to ograniczenie jest największą zaletą zmiennej logicznej — może być tylko prawdziwa lub fałszywa. Jak każda zmienna, także zmienna logiczna wymaga deklaracji przed właściwym użyciem:

```
boolean raining = true; // tworzy zmienną raining i przypisuje jej wartość
                        // początkową true (prawda)
```

Stan (wartość) zmiennej logicznej można zmienić w kodzie szkicu za pomocą prostej operacji przypisania:

```
raining = false;
```

Podejmowanie decyzji na podstawie zmiennych logicznych (za pomocą wyrażeń `if`) jest bardzo proste. Ponieważ wynikiem porównania zmiennych logicznych jest albo wartość `true`, albo wartość `false`, można stosować dla tych zmiennych wprowadzone wcześniej operatory `!=` i `==`. Oto prosty przykład:

```
if ( raining == true )
{
      if ( summer != true )
      {
            // pada deszcz i nie świeci słońce
      }
}
```

Operatory porównania

Za pomocą dostępnych operatorów można podejmować decyzje na podstawie dowolnej liczby zmiennych logicznych lub innych stanów. Programista ma do dyspozycji operatory *negacji* (`!`), *koniunkcji* (`&&`) i *alternatywy* (`||`).

Operator negacji

Operator *negacji* jest reprezentowany przez pojedynczy wykrzyknik (`!`). Operator służy do sprawdzania, czy wartość lub wyrażenie jest fałszywe (*nie jest prawdziwe*). Oto prosty przykład:

```
if ( !raining )
{
        // nie pada deszcz (raining == false)
}
```

Operator koniunkcji

Logiczny operator *koniunkcji* jest reprezentowany przez znaki `&&`. Dzięki temu operatorowi można znacznie ograniczyć liczbę odrębnych wyrażeń `if`. Oto prosty przykład:

```
if (( raining == true ) && ( !summer ))
{
        // pada deszcz i nie świeci słońce (raining == true i summer == false)
}
```

Operator alternatywy

Logiczny operator *alternatywy* jest reprezentowany przez znaki `||`. Stosowanie tego operatora jest bardzo proste; oto przykład:

```
if (( raining == true ) || ( summer == true ))
{
        // pada deszcz lub świeci słońce
}
```

Łączenie wielu operacji porównania

W ramach jednego wyrażenia `if` można stosować wiele operatorów logicznych. Oto prosty przykład:

```
if ( snow == true && rain == true && !hot )
{
        // pada śnieg i deszcz, nie jest gorąco
}
```

Kolejność przetwarzania operacji można określić za pomocą dodatkowych nawiasów. W poniższym przykładzie porównanie zapisane w nawiasach jest przetwarzane jako pierwsze (i zwraca wartość `true` lub `false`), a następnie jego wynik jest łączony z drugim warunkiem w wyrażeniu `if-then`:

```
if (( snow == true || rain == true ) && hot == false))
{
        // pada śnieg lub deszcz, nie jest gorąco
}
```

I wreszcie warto wspomnieć o możliwości testowania wartości zmiennych logicznych bez konieczności stosowania operatorów `== true` lub `== false` w każdym teście (takie rozwiązanie zastosowano już w przykładach użycia operatora negacji). Poniższy fragment kodu działa dokładnie tak samo jak poprzedni przykład:

```
if (( snow || rain ) && !hot )
{
        // pada śnieg lub deszcz, nie jest gorąco
        // (snow ma wartość true LUB rain ma wartość true) ORAZ hot ma wartość false
}
```

Jak widać, system Arduino może podejmować złożone decyzje na podstawie zmiennych logicznych i operatorów porównania. Opisane konstrukcje będą bardzo przydatne podczas realizacji skomplikowanych projektów.

Projekt nr 5: sterowanie ruchem samochodowym

Warto teraz wykorzystać uzyskaną przed chwilą wiedzę do rozwiązania hipotetycznego problemu. Przyjmijmy, że musimy zaplanować ruch wahadłowy na wąskim moście. Na moście dochodzi do jednej – dwóch stłuczek tygodniowo. Zdarzenia zwykle mają miejsce w nocy, kiedy zmęczeni kierowcy zapominają zatrzymać się i sprawdzić, czy można bezpiecznie przejechać przez most. Postanowiliśmy więc zainstalować sygnalizację świetlną, jednak przed podjęciem decyzji o inwestycji przełożony chce zobaczyć prezentację działania systemu. Istnieje oczywiście możliwość instalacji tymczasowej sygnalizacji, jednak koszt takiej demonstracji byłby dość wysoki. Zdecydowaliśmy się więc zbudować model mostu z działającymi sygnalizatorami (w formie diod LED i płytki Arduino).

Cel

Celem projektu jest instalacja sygnalizacji świetlnej (złożonej z trzech świateł różnych kolorów) na obu końcach mostu. Światła spowodują, że ruch w danej chwili będzie się odbywał tylko w jednym kierunku. W momencie wykrycia przez czujniki systemu samochodu oczekującego na wjazd po jednej stronie mostu (stojącego na czerwonym świetle) światło zmienia się, tak aby był możliwy przejazd w odpowiednim kierunku.

Algorytm

Do symulowania czujników pojazdów po obu stronach mostu zostaną użyte dwa przyciski. Każdy sygnalizator świetlny będzie złożony z trzech diod LED: czerwonej, żółtej i zielonej. Początkowo system będzie dopuszczał ruch ze wschodu na zachód, zatem sygnalizator skierowany na wschód będzie pokazywał zielone światło, a sygnalizator skierowany na zachód będzie pokazywał czerwone światło.

Jeśli samochód zbliża się do mostu (do symulowania tego zdarzenia zostanie użyty przycisk) od strony czerwonego światła, system zmieni światło po drugiej stronie mostu z zielonego na czerwone i odczeka pewien czas, aby umożliwić opuszczenie mostu ewentualnym pojazdom, które już się na nim znajdują. Po tym czasie system włączy żółte światło (od strony oczekujących pojazdów), aby dać kierowcom sygnał do przygotowania się przed ostatecznym włączeniem zielonego światła. Światło pozostanie zielone do czasu pojawienia się pojazdów po drugiej stronie. Opisany proces będzie powtarzany bez końca.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- dwie czerwone diody LED (LED1 i LED2);
- dwie żółte diody LED (LED3 i LED4);
- dwie zielone diody LED (LED5 i LED6);
- sześć rezystorów 560 Ω (od R1 do R6);
- dwa rezystory 10 kΩ (R7 i R8);
- dwa kondensatory 100 nF (C1 i C2);
- dwa przyciski (S1 i S2);
- płytka uniwersalna średniej wielkości;
- płytka Arduino i przewód USB;
- różne przewody.

Schemat

Ponieważ system będzie sterował tylko sześcioma diodami LED i będzie otrzymywał sygnały wejściowe od dwóch przycisków, przygotowanie projektu nie będzie zbyt trudne. Schemat obwodu na potrzeby tego projektu pokazano na rysunku 4.26.

Opisywany obwód jest w istocie nieco bardziej rozbudowaną wersją systemu złożonego z przycisku i diody LED stworzonego w ramach projektu nr 4 — tym razem należy zastosować dodatkowe rezystory, więcej diod LED i drugi przycisk.

Diody LED trzeba umieścić na płytce uniwersalnej w odpowiedniej orientacji: rezystory muszą być połączone z anodami diod LED, a katody tych diod muszą być połączone z pinem GND na płytce Arduino (patrz rysunek 4.27).

Rysunek 4.26. Schemat dla projektu nr 5

Rysunek 4.27. Gotowy obwód

Szkic

Czas przygotować właściwy szkic dla tego projektu. Zachęcam do porównania tego kodu z wcześniejszym opisem algorytmu.

```
// Projekt nr 5 — sterowanie ruchem
// definicje pinów, do których podłączono przyciski i diody LED:
❶ #define westButton 7
#define eastButton 13
#define westRed    2
#define westYellow 1
#define westGreen  0
#define eastRed    12
#define eastYellow 11
#define eastGreen  10

#define yellowBlinkTime 500 // żółte światło zapala się na pół sekundy

❷ boolean trafficWest = true; // west = true, east = false
❸ int     flowTime = 10000;    // czas puszczania ruchu pojazdów w jedną stronę
❹ int     changeDelay = 2000; // czas pomiędzy zmianą koloru świateł

void setup()
{
   // ustawia cyfrowe piny wejścia-wyjścia
   pinMode(westButton, INPUT);
   pinMode(eastButton, INPUT);
   pinMode(westRed, OUTPUT);
   pinMode(westYellow, OUTPUT);
   pinMode(westGreen, OUTPUT);
   pinMode(eastRed, OUTPUT);
   pinMode(eastYellow, OUTPUT);
   pinMode(eastGreen, OUTPUT);

   // ustawia początkowy stan świateł (zielone światło od strony zachodniej)
   digitalWrite(westRed, LOW);
   digitalWrite(westYellow, LOW);
   digitalWrite(westGreen, HIGH);
   digitalWrite(eastRed, HIGH);
   digitalWrite(eastYellow, LOW);
   digitalWrite(eastGreen, LOW);
}

void loop()
{
   if ( digitalRead(westButton) == HIGH ) // żądanie puszczenia ruchu
                                          // z zachodu na wschód
   {
      if ( trafficWest != true )
      // ten kod należy wywołać tylko w sytuacji, gdy ruch odbywa się w przeciwnym
      // kierunku (na wschód)
      {
```

```
      trafficWest = true; // zmiana flagi kierunku ruchu na z zachodu
                          // na wschód
      delay(flowTime);    // czas na opuszczenie mostu przez aktualnie
                          // przejeżdżające pojazdy
      digitalWrite(eastGreen, LOW); // zmiana światła od strony
                                    // wschodniej (z zielonego
                                    // na żółte i czerwone)
      digitalWrite(eastYellow, HIGH);
      delay(changeDelay);
      digitalWrite(eastYellow, LOW);
      digitalWrite(eastRed, HIGH);
      delay(changeDelay);
      for ( int a = 0; a < 5; a++ ) // migające żółte światło
      {
         digitalWrite(westYellow, LOW);
         delay(yellowBlinkTime);
         digitalWrite(westYellow, HIGH);
         delay(yellowBlinkTime);
      }
      digitalWrite(westYellow, LOW);
      digitalWrite(westRed, LOW); // zmiana światła od strony zachodniej
                                  // (na zielone)
      digitalWrite(westGreen, HIGH);
   }
}

if ( digitalRead(eastButton) == HIGH ) // żądanie puszczenia ruchu ze
                                       // wschodu na zachód
{
   if ( trafficWest == true )
   // ten kod należy wywołać tylko w sytuacji, gdy ruch odbywa się w przeciwnym
   // kierunku (na zachód)
   {
      trafficWest = false; // zmiana flagi kierunku ruchu na ze wschodu na
                           // zachód
      delay(flowTime);     // czas na opuszczenie mostu przez aktualnie
                           // przejeżdżające pojazdy
      digitalWrite(westGreen, LOW);
      // zmiana światła od strony zachodniej (z zielonego na żółte i czerwone)
      digitalWrite(westYellow, HIGH);
      delay(changeDelay);
      digitalWrite(westYellow, LOW);
      digitalWrite(westRed, HIGH);
      delay(changeDelay);
      for ( int a = 0 ; a < 5 ; a++ ) // migające żółte światło
      {
         digitalWrite(eastYellow, LOW);
         delay(yellowBlinkTime);
         digitalWrite(eastYellow, HIGH);
         delay(yellowBlinkTime);
```

```
        }
        digitalWrite(eastYellow, LOW);
        digitalWrite(eastRed, LOW); // zmiana światła od strony wschodniej
                                    // (z czerwonego na zielone)
        digitalWrite(eastGreen, HIGH);
      }
    }
}
```

Szkic rozpoczyna się od wyrażeń `#define` ❶ nadających odpowiednie etykiety pinom cyfrowym, do których podłączono diody LED i oba przyciski. Po obu stronach mostu (wschodniej i zachodniej) zainstalowano po jednym przycisku i po trzy diody LED: czerwoną, żółtą i zieloną. Zmienna logiczna `trafficWest` ❷ służy do reprezentowania aktualnego kierunku ruchu na moście — ma wartość `true`, jeśli pojazdy przejeżdżają z zachodu na wschód, oraz wartość `false`, jeśli ruch odbywa się ze wschodu na zachód.

UWAGA *Wystarczy jedna zmienna `trafficWest` do reprezentowania kierunku ruchu za pomocą wartości `true` lub `false`. Zastosowanie pojedynczej zmiennej zamiast dwóch (jednaj dla ruchu na wschód i jednej dla ruchu na zachód) eliminuje ryzyko przypadkowego przypisania wartości `true` zmiennym dla obu kierunków (i wypadku wskutek wpuszczenia pojazdów z obu stron)!*

Zmienna całkowitoliczbowa `flowTime` ❸ reprezentuje minimalny czas potrzebny na przejechanie mostu przez samochód. Po włączeniu czerwonego światła system czeka przez ten czas, aby umożliwić opuszczenie mostu pojazdom, które już się na nim znajdują. Zmienna całkowitoliczbowa `changeDelay` ❹ reprezentuje czas pomiędzy samym przełączaniem świateł (z zielonego na żółte, a następnie na czerwone).

Zanim szkic przejdzie do sekcji `void loop()`, w funkcji `void setup()` są wprowadzane ustawienia dla ruchu z zachodu na wschód.

Uruchamianie szkicu

Po uruchomieniu szkic nie podejmuje żadnych działań do momentu naciśnięcia jednego z przycisków. W przypadku wciśnięcia przycisku od strony wschodniej następujący wiersz:

```
if ( trafficWest == true )
```

spowoduje zmianę świateł, pod warunkiem że ruch odbywa się w kierunku przeciwnym do żądanego. Dalsza część kodu jest w istocie prostą sekwencją oczekiwania, włączania i wyłączania różnych diod LED w celu symulacji działania rzeczywistej sygnalizacji świetlnej.

Sygnały analogowe kontra sygnały cyfrowe

W tym podrozdziale wyjaśnię różnice dzielące sygnały cyfrowe od sygnałów analogowych. Omówię też sposób mierzenia sygnałów analogowych za pośrednictwem analogowych pinów wejściowych.

We wszystkich dotychczas opisanych szkicach były stosowane cyfrowe sygnały elektryczne obejmujące dwa odrębne poziomy. W szczególności kod szkiców zawierał wywołania funkcji `digitalWrite(pin, HIGH)` i `digitalWrite(pin, LOW)`, które odpowiednio włączały i wyłączały diodę LED, oraz wywołania funkcji `digitalRead()`, która sprawdzała stan (napięcie) wskazanego pinu — wysoki (`HIGH`) lub niski (`LOW`). Wizualną reprezentację sygnału cyfrowego wielokrotnie zmieniającego stan z wysokiego na niski (i odwrotnie) pokazano na rysunku 4.28.

Rysunek 4.28. Sygnał cyfrowy — stan wysoki (HIGH) jest reprezentowany przez poziome linie na górze; stan niski (LOW) jest reprezentowany przez poziome linie na dole

W przeciwieństwie do sygnałów cyfrowych sygnały analogowe mogą obejmować nieskończenie wiele stanów pomiędzy poziomami niskim i wysokim. Na przykład na rysunku 4.29 pokazano sygnał analogowy w kształcie sinusoidy. Jak widać, napięcie płynnie zmienia się w czasie, osiągając tylko na moment wartości minimalne i maksymalne.

Rysunek 4.29. Sinusoidalny sygnał analogowy

Na platformie Arduino stan wysoki oznacza napięcie bliskie 5 V, natomiast stan niski oznacza napięcie bliskie 0 V (GND). Napięcie sygnału analogowego można mierzyć za pośrednictwem sześciu analogowych pinów wejściowych na płytce Arduino (patrz rysunek 4.30). Piny analogowe mogą służyć do bezpiecznego mierzenia napięć od 0 (GND) do 5 V.

Rysunek 4.30. Analogowe piny wejściowe na płytce Arduino Uno

W kodzie szkicu można użyć funkcji `analog↪Read()`, która zwraca liczbę z przedziału od 0 do 1023 reprezentującą napięcie odczytane za pośrednictwem wskazanego pinu analogowego. W poniższym przykładzie wartość odczytana z pinu analogowego nr 0 i zwrócona przez funkcję `analog↪Read()` jest przypisywana zmiennej całkowitoliczbowej `a`:

```
a = analogRead(0); // odczytuje stan analogowego pinu wejściowego nr 0 (A0)
                   // zwraca wartość od 0 do 1023 (odpowiadającą odpowiednio
                   // napięciu 0,000 i 4,995 V)
```

Projekt nr 6: tester baterii

Mimo że popularność tradycyjnych baterii spada, większość użytkowników nadal posługuje się w domu urządzeniami zasilanymi bateriami AA, AAA, C lub D, na przykład pilotami do telewizorów, zegarami czy zabawkami dla dzieci. Baterie oferują dużo niższe napięcie niż 5 V, zatem możemy wykorzystać system Arduino do mierzenia dostarczanego napięcia i — tym samym — stanu sprawdzanej baterii. W ramach tego projektu zbudujemy prosty tester baterii.

Cel

Napięcie dostarczane przez typową nową baterię AA zwykle wynosi około 1,6 V i z czasem spada wraz ze zużyciem baterii. System budowany w ramach tego projektu będzie mierzył napięcie i prezentował stan baterii za pomocą diod LED. Stan baterii będzie odczytywany za pomocą funkcji `analogRead()` i konwertowany na wartość wyrażoną w woltach. Ponieważ maksymalne napięcie obsługiwane przez analogowy pin wejściowy wynosi 5 V, należy podzielić 5 przez 1024 (czyli liczbę możliwych wartości) — w ten sposób otrzymujemy wartość 0,0048. Oznacza to, że jeśli funkcja `analogRead()` zwróci na przykład wartość 512, należy pomnożyć tę liczbę przez 0,0048, zatem wskazanie pinu analogowego wyniesie 2,4576 V.

Algorytm

Oto algorytm przygotowany dla systemu testera baterii:

1. Odczytaj stan pinu analogowego nr 0.

2. Pomnóż odczytaną wartość przez 0,0048, aby uzyskać wartość napięcia.
3. Jeśli napięcie jest większe lub równe 1,6 V, włącz na krótko zieloną diodę LED.
4. Jeśli napięcie jest większe niż 1,4 V, ale mniejsze niż 1,6 V, włącz na krótko żółtą diodę LED.
5. Jeśli napięcie jest mniejsze niż 1,4 V, włącz na krótko czerwoną diodę LED.
6. Powtarzaj ten proces w nieskończoność.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- trzy rezystory 560 (od R1 do R3);
- jeden rezystor 2,2 kΩ (R4);
- zielona dioda LED (LED1);
- żółta dioda LED (LED2);
- czerwona dioda LED (LED3);
- płytka uniwersalna;
- różne przewody;
- płytka Arduino i przewód USB.

Schemat

Schemat obwodu dla tego prostego testera baterii pokazano na rysunku 4.31. Warto zwrócić uwagę na dwie końcówki po lewej stronie oznaczone znakami + i –. Właśnie do tych końcówek należy podłączyć *odpowiednie* strony testowanej baterii. Stronę dodatnią należy połączyć z końcówką oznaczoną plusem, natomiast stronę ujemną należy połączyć z końcówką oznaczoną minusem.

OSTRZEŻENIE *W żadnym razie nie wolno podejmować prób używania tego obwodu do mierzenia napięcia przekraczającego 5 V. Nie należy też łączyć strony dodatniej ze stroną ujemną (i odwrotnie). Każda taka próba spowoduje uszkodzenie płytki Arduino.*

Szkic

Oto szkic dla tego projektu:

```
// Projekt nr 6 — tester baterii
#define newLED 2 // zielona dioda LED (nowa bateria)
#define okLED 4  // żółta dioda LED (sprawna bateria)
#define oldLED 6 // czerwona dioda LED (zużyta bateria)

int analogValue = 0;
❶ float voltage = 0;
```

Rysunek 4.31. Schemat dla projektu nr 6

```
int ledDelay = 2000;

void setup()
{
   pinMode(newLED, OUTPUT);
   pinMode(okLED, OUTPUT);
   pinMode(oldLED, OUTPUT);
}

void loop()
{
❷     analogValue = analogRead(0);
❸     voltage = 0.0048*analogValue;
❹     if ( voltage >= 1.6 )
      {
         digitalWrite(newLED, HIGH);
         delay(ledDelay);
         digitalWrite(newLED, LOW);
      }
❺        else if ( voltage < 1.6 && voltage > 1.4 )
      {
         digitalWrite(okLED, HIGH);
         delay(ledDelay);
         digitalWrite(okLED, LOW);
      }
```

```
❻       else if ( voltage <= 1.4 )
    {
       digitalWrite(oldLED, HIGH);
       delay(ledDelay);
       digitalWrite(oldLED, LOW);
    }
}
```

W szkicu dla projektu nr 6 system Arduino odczytuje wartość zmierzoną przez pin analogowy nr 0 ❷ i konwertuje ją na wartość reprezentującą rzeczywiste napięcie ❸. W szkicu zastosowano też nowy typ zmiennej, `float` ❶, który zostanie omówiony w następnym podrozdziale. W kodzie zastosowano też znane konstrukcje, jak wyrażenia `if-else`, oraz kilka nowych elementów, w tym operacje arytmetyczne i operatory porównujące wartości liczbowe. Wszystkie te nowe elementy zostaną opisane w następnym i kolejnych podrozdziałach.

Działania arytmetyczne w systemie Arduino

System Arduino może być używany do zadań typowych dla kieszonkowych kalkulatorów i może wykonywać na przykład operacje mnożenia, dzielenia, dodawania i odejmowania. Oto kilka przykładów działań zapisanych w kodzie:

```
a = 100;
b = a + 20;
c = b - 200;
d = c + 80; // d będzie równe 0
```

Zmienne typu float

W sytuacji gdy program musi operować na liczbach z punktem dziesiętnym (i częścią ułamkową), wystarczy zastosować zmienną typu `float`. Wartości przechowywane w zmiennych tego typu mieszczą się w przedziale od $-3{,}4028235 \times 10^{38}$ do $3{,}4028235 \times 10^{38}$, a ich precyzja najczęściej jest ograniczona do sześciu lub siedmiu miejsc po przecinku. W obliczeniach można dowolnie łączyć liczby całkowite z liczbami zmiennoprzecinkowymi. Szkic może na przykład zawierać operację dodawania liczby zmiennoprzecinkowej `f` do liczby całkowitej `a`, przypisującą wynik zmiennej `g` typu `float`:

```
int a = 100;
float f;
float g;
f = a / 3; // f = 33,333333
g = a + f; // g = 133,333333
```

Operatory porównania liczb

Podstawowe operatory porównania, czyli `==` i `!=`, były już stosowane w wyrażeniach `if` do obsługi cyfrowych sygnałów wejściowych w projekcie nr 5. Oprócz wymienionych operatorów istnieją jeszcze następujące operatory umożliwiające porównywanie liczb i zmiennych numerycznych:

- `<` mniejszy niż
- `>` większy niż
- `<=` mniejszy lub równy
- `>=` większy lub równy

Wymienione operatory były już używane do porównywania liczb w szkicu dla projektu nr 6, w wierszach ❹, ❺ i ❻.

Poprawa precyzji pomiarów sygnału analogowego za pomocą napięcia referencyjnego

Jak wspomniałem przy okazji omawiania projektu nr 6, funkcja `analogRead()` zwraca wartość proporcjonalną do odczytanego napięcia, które może wynosić od 0 do 5 V. Górna granica (5 V) to tzw. *napięcie referencyjne* (ang. *reference voltage*), czyli maksymalne napięcie akceptowane na pinie analogowym płytki Arduino, dla którego funkcja `analogRead()` zwraca najwyższą możliwą wartość (równą 1023).

Aby podnieść precyzję pomiarów niższych napięć, należy obniżyć napięcie referencyjne. Jeśli napięcie referencyjne wynosi 5 V, funkcja `analogRead()` reprezentuje odczytany poziom za pomocą wartości od 0 do 1023. Gdyby jednak budowany system miał służyć na przykład do mierzenia napięcia, które nie przekracza 2 V, i gdyby także ten przedział napięcia był reprezentowany przez wartości z przedziału od 0 do 1023, precyzja pomiarów byłaby dużo wyższa. Okazuje się, że można to osiągnąć, stosując zewnętrzne lub wewnętrzne napięcie referencyjne (obie techniki zostaną omówione w poniższych punktach).

Stosowanie zewnętrznego napięcia referencyjnego

Rysunek 4.32. Pin AREF na płytce Arduino Uno

Pierwsza metoda stosowania napięcia referencyjnego polega na użyciu pinu analogowego AREF (od ang. *analog reference*); patrz rysunek 4.32.

Nowe napięcie referencyjne można wprowadzić, łącząc odpowiednie źródło napięcia z pinami AREF i GND na płytce Arduino. W ten sposób można obniżyć napięcie referencyjne, ale nie można go podnieść, ponieważ napięcie referencyjne podłączone do płytki Arduino Uno nigdy nie może przekraczać 5 V. Jednym z najprostszych sposobów

obniżenia napięcia referencyjnego jest utworzenie *dzielnika napięcia* złożonego z dwóch rezystorów (patrz rysunek 4.33).

Rysunek 4.33. Obwód dzielnika napięcia

Wartości *R1* i *R2* mają wpływ na napięcie referencyjne zgodnie z poniższym wzorem:

$$V_{wy} = V_{we}\left(\frac{R2}{R1 + R3}\right)$$

V_{wy} to napięcie referencyjne, V_{we} to napięcie wejściowe (w tym przypadku 5 V). *R1* i *R2* to wartości oporności obu rezystorów wyrażone w omach.

Najprostszym sposobem dzielenia napięcia jest dwukrotne obniżenie V_{we} poprzez zastosowanie dwóch takich samych oporników *R1* i *R2*, na przykład o wartości 10 kΩ każdy. W takim przypadku warto zastosować rezystory z możliwie najniższym poziomem tolerancji (na przykład 1%) oraz sprawdzić rzeczywiste wartości rezystancji przy użyciu multimetru, a następnie potwierdzić wszystkie parametry za pomocą przytoczonego wzoru. Warto też umieścić kondensator 100 nF pomiędzy pinami AREF i GND, aby uniknąć zakłóceń na pinie AREF, które mogą powodować niestabilne wskazania analogowego pinu wejściowego.

W przypadku zastosowania zewnętrznego napięcia referencyjnego należy umieścić w funkcji `void setup()` w szkicu następujący wiersz:

```
analogReference(EXTERNAL); // wskazuje pin AREF jako źródło napięcia referencyjnego
```

Stosowanie wewnętrznego napięcia referencyjnego

Płytka Arduino Uno oferuje także wewnętrzne napięcie referencyjne na poziomie 1,1 V. Jeśli ten poziom napięcia jest wystarczający, nie ma potrzeby stoso-

wania żadnych dodatkowych urządzeń. Wystarczy umieścić w ciele funkcji `void setup()` następujący kod:

```
analogReference(INTERNAL); // wskazuje wewnętrzne źródło napięcia 1,1 V jako
                           // napięcie referencyjne
```

Rezystor nastawny

Rezystory nastawne, znane też jako *potencjometry*, umożliwiają zmianę wartości rezystancji od 0 Ω do pewnej wartości maksymalnej. Symbol rezystora nastawnego pokazano na rysunku 4.34.

Rysunek 4.34. Symbol rezystora nastawnego (potencjometru)

Rezystory nastawne mają trzy piny — jeden w środku i dwa po bokach. Obrót gałki rezystora nastawnego powoduje zwiększenie oporu elektrycznego pomiędzy jedną stroną a środkowym pinem oraz zmniejszenie oporu elektrycznego pomiędzy drugą stroną a środkowym pinem.

Rezystory nastawne są dostępne w wersjach *liniowych* i *logarytmicznych*. W potencjometrach liniowych obrót gałki powoduje zmianę oporu ze stałą szybkością, natomiast w modelach logarytmicznych zmiana początkowo jest niewielka, ale dalszy obrót powoduje błyskawiczny wzrost lub spadek wartości rezystancji. Potencjometry logarytmiczne zwykle stosuje się we wzmacniaczach dźwięku, ponieważ lepiej sprawdzają się podczas dostosowywania parametrów do ludzkiego słuchu. W większości projektów na bazie płytki Arduino stosuje się liniowe rezystory nastawne podobne do tego z rysunku 4.35.

Rysunek 4.35. Typowy liniowy rezystor nastawny

Istnieją też miniaturowe wersje rezystorów nastawnych znane jako *trimpoty* lub *trymery* (patrz rysunek 4.36). Z uwagi na niewielkie rozmiary trimpoty stosuje się głównie do regulacji parametrów w ramach obwodów. Ważną zaletą tych miniaturowych potencjometrów jest możliwość ich instalacji na płytkach uniwersalnych.

Rysunek 4.36. Przykłady różnych trimpotów

UWAGA *Przed zakupem trimpotów warto sprawdzić ich typ. W wielu przypadkach najlepszym rozwiązaniem jest trimpot regulowany śrubokrętem i zamknięty w odpowiedniej obudowie (jak na rysunku 4.36), ponieważ zwykle jest dużo trwalszy niż tańsze, otwarte modele.*

Brzęczyki piezoelektryczne

Brzęczyk piezoelektryczny (w skrócie *piezo*) to niewielkie, okrągłe urządzenie generujące głośny, dość irytujący dźwięk. Urządzenia tego typu doskonale sprawdzają się w przypadku alarmów lub podczas budowy rozmaitych projektów. Przykład typowego brzęczyka, TDK PS1240, pokazano na rysunku 4.37 (widoczna obok moneta ćwierćdolarowa dobrze ilustruje niewielki rozmiar urządzenia).

Rysunek 4.37. Brzęczyk piezo TDK PS1240

Brzęczyki piezoelektryczne zawierają bardzo cienką płytkę wewnątrz obudowy, która porusza się po włączeniu zasilania. Zasilanie brzęczyka prądem przemiennym (na przykład sekwencją włączony, wyłączony, włączony, wyłączony, ...) powoduje wibracje wspomnianej płytki i generowanie dźwięku.

Stosowanie tych urządzeń łącznie z płytką Arduino jest bardzo proste, ponieważ sygnał można włączać i wyłączać dokładnie tak jak w przypadku diod LED. Elementy piezo nie są spolaryzowane i jako takie mogą być łączone z obwodem w dowolnej orientacji.

Schemat elementu piezo

Symbol brzęczyka piezoelektrycznego stosowany na schematach obwodów przypomina rysunek głośnika (patrz rysunek 4.38), zatem rozpoznanie tego urządzenia nie stanowi żadnego problemu.

Rysunek 4.38. Symbol brzęczyka piezoelektrycznego

UWAGA *Podczas wyboru brzęczyka piezoelektrycznego na potrzeby opisanego poniżej projektu warto zwrócić uwagę na rodzaj urządzenia — niektóre typy brzęczyków wyglądają dokładnie tak jak ten pokazany na rysunku 4.38, ale obejmują dodatkowe wbudowane obwody odpowiedzialne za generowanie różnych tonów. W tym przypadku dodatkowe obwody nie są konieczne, ponieważ za wysokość dźwięku będzie odpowiadał bezpośrednio system Arduino.*

Projekt nr 7: praktyczne wykorzystanie brzęczyka piezo

Każdy, kto dysponuje brzęczykiem piezoelektrycznym, może od razu wysłać poniższy szkic na płytkę Arduino i sprawdzić, jak ten brzęczyk działa w praktyce:

```
// Projekt nr 7 — praktyczne wykorzystanie brzęczyka piezo
#define PIEZO 3 // pin nr 3 umożliwia wysyłanie sygnału generującego dźwięk
                // różnych tonów
int del = 500;
void setup()
{
   pinMode(PIEZO, OUTPUT);
}

void loop()
{
❶ analogWrite(PIEZO, 128);  // 50% współczynniki wypełnienia sygnału
                            // wysyłanego do brzęczyka
   delay(del);
   digitalWrite(PIEZO, LOW); // wyłącza brzęczyk
   delay(del);
}
```

W szkicu zastosowano technikę modulacji szerokości impulsu wysyłanego za pośrednictwem pinu nr 3. Zmiana współczynnika wypełnienia przekazanego na wejściu funkcji `analogWrite()` (wartość 128 reprezentuje współczynnik na poziomie 50%) ❶ wpływa na głośność dźwięku emitowanego przez brzęczyk.

Aby zwiększyć głośność brzęczyka, należy podnieść napięcie zasilania tego urządzenia. Napięcie jest obecnie ograniczone do 5 V, jednak zasilanie prądem o napięciu 9 lub 12 V pozwoliłoby generować dużo głośniejszy dźwięk. Ponieważ jednak płytka Arduino nie może dostarczać napięcia wyższego niż 5 V, należałoby zastosować dla brzęczyka zewnętrzne źródło zasilania, na przykład baterię 9 V, i sterować zasilaniem brzęczyka za pomocą tranzystora. Dla schematu pokazanego na rysunku 4.39 można zastosować dokładnie ten sam szkic.

Rysunek 4.39. Schemat dla projektu nr 7

Część schematu oznaczona wartością 12 V reprezentuje dodatnią stronę źródła zasilania (ujemną stronę należy połączyć z pinem GND na płytce Arduino).

Projekt nr 8: budowa szybkiego termometru

Temperatura może być reprezentowana przez sygnał analogowy. Temperaturę można mierzyć na przykład za pomocą czujnika TMP36 firmy Analog Devices (*http://www.analog.com/tmp36/*), który pokazano na rysunku 4.40.

Rysunek 4.40. Czujnik temperatury TMP36

Czujnik TMP36 wygląda bardzo podobnie do tranzystora BC548, który zastosowano w obwodzie opisanym w rozdziale 3. Czujnik TMP36 przekazuje na wyjściu napięcie proporcjonalne do temperatury otoczenia, zatem określenie bieżącej temperatury wymaga tylko prostej konwersji odczytywanej wartości. Na przykład temperatura 25 stopni Celsjusza jest reprezentowana przez prąd o napięciu 750 mV, a każda zmiana temperatury o 1 stopień odpowiada zmianie napięcia o 10 mV. Czujnik TMP36 może służyć do mierzenia temperatur z przedziału od −40 do 125 stopni Celsjusza.

Funkcja `analogRead()` zwraca wartość z przedziału od 0 do 1023, która odpowiada napięciu z przedziału od 0 do nieco poniżej 5000 mV (5 V). Wystarczy pomnożyć wartość zwracaną przez funkcję `analogRead()` przez (5000 : 1024), aby uzyskać rzeczywiste napięcie przekazywane przez czujnik. Należy jeszcze odjąć wartość 500 (czyli zastosowane w czujniku TMP36 przesunięcie niezbędne do reprezentowania temperatur poniżej zera) i podzielić otrzymaną wartość przez 10 — uzyskana w ten sposób liczba będzie reprezentowała temperaturę w stopniach Celsjusza. Aby otrzymać temperaturę w stopniach Fahrenheita, należy pomnożyć liczbę stopni Celsjusza przez 1,8 i dodać do wyniku liczbę 32.

Cel

W tym projekcie zostanie użyty czujnik TMP36 do szybkiego sprawdzania temperatury. Jeśli temperatura spada poniżej 20 stopni Celsjusza, jest włączana niebieska dioda LED. Jeśli temperatura mieści się w przedziale od 20 do 26 stopni, jest włączana zielona dioda LED, a jeśli temperatura przekracza 26 stopni, system włącza czerwoną diodę LED.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- trzy rezystory 560 Ω (od R1 do R3);
- czerwona dioda LED (LED1);
- zielona dioda LED (LED2);
- niebieska dioda LED (LED3);
- czujnik temperatury TMP36;
- płytka uniwersalna;

- różne przewody;
- płytka Arduino i przewód USB.

Schemat

Obwód potrzebny do realizacji tego projektu jest dość prosty. Jeśli czujnik TMP36 jest ustawiony w taki sposób, że jest widoczna nadrukowana etykieta, pin po lewej stronie to wejście 5 V, pin po środku to wyjście, a pin po prawej stronie służy do połączenia z uziemieniem (patrz rysunek 4.41).

Rysunek 4.41. Schemat dla projektu nr 8

Szkic

Czas przygotować szkic dla tego projektu:

```
// Projekt nr 8 — budowa szybkiego termometru

// definicje pinów, do których podłączono diody LED:
#define HOT 6
#define NORMAL 4
#define COLD 2

float voltage = 0;
float celsius = 0;
```

```
float hotTemp = 26;
float coldTemp = 20;
float sensor = 0;

void setup()
{
   pinMode(HOT, OUTPUT);
   pinMode(NORMAL, OUTPUT);
   pinMode(COLD, OUTPUT);
}

void loop()
{
   // odczytuje stan czujnika temperatury i konwertuje wynik na stopnie Celsjusza
❶ sensor = analogRead(0);
   voltage = (sensor*5000)/1024; // konwertuje wskazanie czujnika na miliwolty
   voltage = voltage-500;        // odejmuje wartość przesunięcia
   celsius = voltage/10;         // konwertuje miliwolty na stopnie Celsjusza

   // obsługa przedziałów temperatur:
❷ if ( celsius < coldTemp )
   {
      digitalWrite(COLD, HIGH);
      delay(1000);
      digitalWrite(COLD, LOW);
   }
❸ else if ( celsius > coldTemp && celsius <= hotTemp )
   {
      digitalWrite(NORMAL, HIGH);
      delay(1000);
      digitalWrite(NORMAL, LOW);
   }
   else
   {
      // celsius > hotTemp
      digitalWrite(HOT, HIGH);
      delay(1000);
      digitalWrite(HOT, LOW);
   }
}
```

Szkic w tej formie odczytuje wartość napięcia z czujnika TMP36 i konwertuje ją na temperaturę wyrażoną w stopniach Celsjusza ❶. Wyrażenia `if-else` w wierszach ❷ i ❸ porównują bieżącą temperaturę z wartościami progowymi dla wysokiej i niskiej temperatury, aby na tej podstawie system mógł włączyć odpowiednią diodę LED. Wyrażenia `delay(1000)` zapobiegają zbyt szybkiemu miganiu diod LED w sytuacji, gdy temperatura jest zbliżona do wartości progowej i jest w krótkim czasie wielokrotnie, naprzemiennie przypisywana do dwóch przedziałów.

Doskonalenie szkicu

Mimo że opisany szkic jest dość prosty, warto go użyć jako punktu wyjścia do realizacji innych projektów polegających na odczytywaniu danych z czujników. System można uzupełnić na przykład o przełącznik PowerSwitch Tail (patrz rysunek 4.42).

Rysunek 4.42. Przełącznik PowerSwitch Tail umożliwiający włączanie i wyłączanie zasilania prądu przemiennego o maksymalnym napięciu 120 V

Za pomocą przełącznika PowerSwitch Tail podłączonego do wyjścia cyfrowego płytki Arduino można bezpiecznie sterować urządzeniami zasilanymi z gniazd sieciowych, na przykład grzejnikami lub lampami. (Więcej informacji na temat tego przełącznika można znaleźć na stronie *http://www.adafruit.com/products/268/*). Przy użyciu przełącznika PowerSwitch Tail można zbudować na przykład automatycznie sterowany (zależnie od temperatury) grzejnik lub wentylator, oświetlenie garażowe włączające się na pewien czas lub zdalnie sterowane, zewnętrzne lampki świąteczne.

Co dalej?

Rozdział 4. zbliża się do końca. Wprowadziłem mnóstwo nowych narzędzi, w tym cyfrowe piny wejściowe i wyjściowe, nowe rodzaje zmiennych i kilka przydatnych funkcji matematycznych. W następnym rozdziale omówię m.in. nowe, jeszcze ciekawsze techniki wykorzystania diod LED, opiszę sposoby tworzenia własnych funkcji, przygotuję grę komputerową oraz elektroniczną kostkę do gry i wiele więcej.

5

Praca z funkcjami

W tym rozdziale:

- utworzysz własne funkcje;
- nauczysz się podejmowania decyzji w kodzie za pomocą konstrukcji `while` i `do-while`;
- opanujesz wysyłanie i odbieranie danych pomiędzy płytką Arduino a oknem monitora portu szeregowego;
- poznasz zmienne typu `long`.

W tym rozdziale wprowadzę techniki tworzenia własnych funkcji, czyli konstrukcji znacznie poprawiających czytelność szkiców dla platformy Arduino i upraszczających budowane programy. Pokażę też, jak tworzyć kod podzielony na moduły wielokrotnego użytku, dzięki którym można oszczędzić mnóstwo czasu. Omówię techniki podejmowania decyzji wpływających na wykonywanie bloków kodu. Wprowadzę też nowy typ zmiennych całkowitoliczbowych nazwany `long`. W dalszej części tego rozdziału przygotuję nowy system termometru przy użyciu własnych, niestandardowych funkcji.

Funkcja składa się ze zbioru instrukcji, które można stosować w dowolnym miejscu szkicu. Mimo że język Arduino oferuje wiele gotowych funkcji, w pewnych sytuacjach trudno znaleźć rozwiązanie w pełni odpowiadające potrzebom programisty. Zdarza się też, że pewien fragment szkicu musi być wykonywany

wielokrotnie (każdorazowe powtarzanie fragmentu kodu byłoby marnotrawstwem pamięci). W obu przypadkach brakującym elementem jest wyspecjalizowana funkcja realizująca konkretne zadania. Okazuje się, że taka funkcja istnieje — każdy programista może ją sam utworzyć.

Projekt nr 9: tworzenie funkcji powtarzającej określone działanie

Za pomocą prostych funkcji można łatwo (na żądanie) powtarzać pewne działania. Na przykład poniższa funkcja dwukrotnie włącza i wyłącza wbudowaną diodę LED (odpowiednio w wierszach ❶ i ❸ oraz ❷ i ❹).

```
void blinkLED()
{
   ❶ digitalWrite(13, HIGH);
     delay(1000);
   ❷ digitalWrite(13, LOW);
     delay(1000);
   ❸ digitalWrite(13, HIGH);
     delay(1000);
   ❹ digitalWrite(13, LOW);
     delay(1000);
}
```

Oto przykład użycia tej funkcji w ramach gotowego szkicu, który w tej formie można wysłać na płytkę Arduino:

```
// Projekt nr 9 — tworzenie funkcji powtarzającej określone działanie
#define LED 13
#define del 200

void setup()
{
   pinMode(LED, OUTPUT);
}

void blinkLED()
{
   digitalWrite(LED, HIGH);
   delay(del);
   digitalWrite(LED, LOW);
   delay(del);
   digitalWrite(LED, HIGH);
   delay(del);
   digitalWrite(LED, LOW);
```

```
   delay(del);
}

void loop()
{
❶ blinkLED();
   delay(1000);
}
```

Wywołanie funkcji `blinkLED()` w ciele funkcji `void loop()` ❶ spowoduje, że system Arduino wykona polecenia zawarte w sekcji `void blinkLED()`. Innymi słowy, `blinkLED()` jest funkcją, którą można stosować w dowolnym miejscu kodu.

Projekt nr 10: tworzenie funkcji ustawiającej liczbę cykli włączania diod

Funkcja utworzona w poprzednim podrozdziale oferuje dość ograniczone możliwości. Co należałoby zrobić, aby zmienić liczbę cykli migania diody i czas opóźnienia? Okazuje się, że to żaden problem — wystarczy utworzyć funkcję umożliwiającą modyfikację tych wartości:

```
void blinkLED(int cycles, int del)
{
   for ( int z = 0 ; z < cycles ; z++ )
   {
      digitalWrite(LED, HIGH);
      delay(del);
      digitalWrite(LED, LOW);
      delay(del);
   }
}
```

Nowa funkcja `void blinkLED()` otrzymuje na wejściu dwie wartości całkowitoliczbowe: `cycles` (liczbę cykli migania diody LED) oraz `del` (czas oczekiwania pomiędzy włączaniem a wyłączaniem diody LED). Jeśli więc dioda LED ma być włączona i wyłączona 12 razy i jeśli czas świecenia diody w każdym cyklu ma wynosić 100 milisekund, należy zastosować wywołanie `blinkLED(12, 100)`. Aby sprawdzić działanie tej funkcji, wystarczy wpisać w środowisku IDE następujący szkic:

```
// Projekt nr 10 — tworzenie funkcji ustawiającej liczbę cykli włączania diod

#define LED 13
```

```
void setup()
{
   pinMode(LED, OUTPUT);
}

void blinkLED(int cycles, int del)
{
   for ( int z = 0 ; z < cycles ; z++ )
   {
      digitalWrite(LED, HIGH);
      delay(del);
      digitalWrite(LED, LOW);
      delay(del);
   }
}

void loop()
{
❶ blinkLED(12, 100);
   delay(1000);
}
```

W wierszu ❶ wartości 12 i 100 (określające odpowiednio liczbę cykli i opóźnienie) są przekazywane do funkcji `blinkLED()`, w której po tym wywołaniu zmienna `cycles` będzie miała wartość 12, a zmienna `del` będzie równa 100. Oznacza to, że dioda LED zostanie włączona i wyłączona 12 razy, a pomiędzy kolejnymi cyklami minie 100 milisekund.

Tworzenie funkcji zwracającej wartość

Oprócz tworzenia funkcji, które otrzymują na wejściu wartości w formie parametrów (tak jak funkcja `void blinkLED()` z projektu nr 10), istnieje możliwość tworzenia funkcji zwracających wartości (tak jak funkcja `analogRead()`, zwracająca wartości z przedziału od 0 do 1023 reprezentujące pomiar na analogowym pinie wejściowym). Słowo `void` zapisywane przed nazwami dotychczas tworzonych funkcji oznaczało, że te funkcje niczego nie zwracały — typem zwracanych wartości był `void`, czyli brak wartości. Warto teraz spróbować opracować kilka przydatnych funkcji, które będą zwracały konkretne wartości.

Poniższy kod implementuje funkcję konwertującą stopnie Celsjusza na stopnie Fahrenheita:

```
float convertTemp(float celsius)
{
   float fahrenheit = 0;
   fahrenheit = (1.8 * celsius) + 32;
   return fahrenheit;
}
```

W pierwszym wierszu zdefiniowano nazwę funkcji (w tym przypadku `convertTemp`), typ zwracanej wartości (`float`) oraz wszystkie zmienne, które mają być przekazywane na wejściu funkcji (`float celsius`). Aby użyć tej funkcji, należy wskazać istniejącą zmienną, której zostanie przypisana wartość wynikowa. Żeby na przykład przekonwertować 40 stopni Celsjusza na stopnie Fahrenheita i zapisać wynik w zmiennej `tempf` typu `float`, wystarczy użyć wywołania funkcji `convertTemp` w następującej formie:

```
tempf = convertTemp(40);
```

Wartość 40 zostanie przypisana zmiennej `celsius` w ramach funkcji `convertTemp` i wykorzystana w wyrażeniu `fahrenheit = (1.8 * celsius) + 32`. Wiersz `return fahrenheit` w kodzie funkcji `convertTemp` powoduje zwrócenie otrzymanej wartości (która zostanie następnie przypisana zmiennej `tempf`).

Projekt nr 11: budowa szybkiego termometru z migającymi diodami LED

Skoro wyjaśniłem już, jak można tworzyć niestandardowe funkcje, czas wykorzystać tę wiedzę do przebudowy projektu termometru z rozdziału 4., czyli systemu obejmującego czujnik temperatury TMP36 i wbudowaną diodę LED na płytce Arduino. Jeśli temperatura spada poniżej 20 stopni Celsjusza, dioda LED miga dwukrotnie, po czym gaśnie; jeśli temperatura mieści się w przedziale od 20 do 26 stopni Celsjusza, dioda LED miga czterokrotnie; jeśli temperatura wzrasta powyżej 26 stopni Celsjusza, dioda LED miga sześciokrotnie.

Nowy szkic będzie bardziej modułowy dzięki podziałowi kodu na odrębne funkcje — tak przygotowany program będzie łatwiejszy do zrozumienia, a same funkcje będzie można wykorzystać w innych szkicach. Sam termometr będzie realizował dwa zadania: będzie mierzył temperaturę i przypisywał ją do odpowiedniej kategorii oraz generował sygnał polegający na miganiu diody LED określoną liczbę razy (zależnie od temperatury).

Sprzęt

Wymagania sprzętowe tego projektu są minimalne:

- czujnik temperatury TMP36;
- płytka uniwersalna;
- różne przewody;
- płytka Arduino i przewód USB.

Schemat

Obwód niezbędny do budowy tego projektu jest bardzo prosty (schemat obwodu pokazano na rysunku 5.1).

Rysunek 5.1. Schemat obwodu dla projektu nr 11

Szkic

Na potrzeby tego projektu należy utworzyć dwie funkcje w ramach szkicu. Pierwsza będzie odczytywała wartość z czujnika TMP36, konwertowała ją na stopnie Celsjusza, a następnie zwracała liczbę 2, 4 lub 6 (określającą liczbę cykli migania diody LED). Szkic dla tego projektu zostanie utworzony na bazie programu dla projektu nr 8.

W roli drugiej funkcji zostanie użyta gotowa funkcja `blinkLed()` z projektu nr 9. Funkcja `void loop()` będzie wywoływała funkcję `blinkLed()`, a następnie wstrzymywała działanie na dwie sekundy przed rozpoczęciem kolejnego cyklu.

UWAGA *Należy pamiętać o konieczności zapisywania zmodyfikowanych szkiców projektów i nadawania im nowych nazw plików, by przypadkowo nie nadpisać istniejących projektów!*

Aby utworzyć nowy szkic, należy wpisać następujący kod w środowisku IDE:

```
// Projekt nr 11 — budowa szybkiego termometru z migającymi diodami LED

#define LED 13

int blinks = 0;

void setup()
{
   pinMode(LED, OUTPUT);
}

int checkTemp()
{
   float voltage = 0;
   float celsius = 0;
   float hotTemp = 26;
   float coldTemp = 20;
   float sensor = 0;
   int result;

   // odczytuje stan czujnika temperatury i konwertuje wynik na stopnie Celsjusza
   sensor = analogRead(0);
   voltage = (sensor * 5000) / 1024; // konwertuje wskazanie czujnika na miliwolty
   voltage = voltage - 500;        // odejmuje przesunięcie stosowane dla napięcia
   celsius = voltage / 10;        // konwertuje miliwolty na stopnie Celsjusza

   // obsługuje przedziały temperatur
   if (celsius < coldTemp)
   {
      result = 2;
   }
   else if (celsius >= coldTemp && celsius <= hotTemp)
   {
      result = 4;
   }
   else
   {
      result = 6;  // (celsius > hotTemp)
   }
   return result;
}
void blinkLED(int cycles, int del)
{
for ( int z = 0 ; z < cycles ; z++ )
{
digitalWrite(LED, HIGH);
delay(del);
digitalWrite(LED, LOW);
delay(del);
}
```

```
}
void loop()
{
blinks = checkTemp();
blinkLED(blinks, 500);
delay(2000);
}
```

Ponieważ w szkicu zastosowano niestandardowe funkcje, w ciele funkcji `void loop()` wystarczy wywołać te funkcje i ustawić czas oczekiwania. Funkcja `check ➥Temp()` zwraca wartość całkowitoliczbową, która jest przypisywana zmiennej `blinks`. Funkcja `blinkLED()` odpowiada za miganie diody LED tyle razy, ile wynosi wartość zmiennej `blinks` (z opóźnieniem 500 milisekund). Przed powtórzeniem całej procedury szkic wstrzymuje działanie na 2 sekundy.

Wystarczy teraz wysłać szkic na płytkę Arduino i obserwować diodę LED migającą zależnie od temperatury otoczenia. (Nie należy jeszcze rozłączać tego obwodu, ponieważ będzie potrzebny w następnych przykładach).

Wyświetlanie danych odbieranych od płytki Arduino w oknie monitora portu szeregowego

Do tej pory wszystkie szkice były wysyłane na płytkę Arduino, a do prezentowania danych wyjściowych służyły diody LED (ten sposób wyświetlania był stosowany zarówno dla temperatury, jak i dla sygnalizacji świetlnej). Migające diody LED mogą co prawda stanowić wygodne źródło wiedzy o stanie systemu Arduino, ale z pewnością nie wystarczą do przekazywania bardziej złożonych komunikatów. W tym podrozdziale wyjaśnię, jak używać połączenia przewodowego z systemem Arduino i okna monitora portu szeregowego dostępnego w środowisku IDE do wyświetlania danych przekazywanych przez płytkę Arduino do komputera i wysyłania na tę płytkę danych wprowadzanych za pomocą klawiatury.

Monitor portu szeregowego

Aby otworzyć monitor portu szeregowego, należy uruchomić środowisko IDE i kliknąć ikonę *Monitor portu szeregowego* na pasku narzędzi (patrz rysunek 5.2). Środowisko IDE powinno otworzyć okno monitora portu szeregowego podobne do tego z rysunku 5.3.

Rysunek 5.2. Ikona Monitor portu szeregowego na pasku narzędzi środowiska IDE

Jak widać na rysunku 5.3, monitor portu szeregowego obejmuje pole wejściowe w górnej części okna (złożone z pojedynczego wiersza i przycisku *Send*) oraz wielowierszowe pole wyjściowe dla danych przekazywanych przez płytkę Arduino. Zaznaczenie pola wyboru *Automatyczne przewijanie* powoduje, że w dolnym polu będą wyświetlane najnowsze dane wyjściowe (starsze dane będą

Rysunek 5.3. Monitor portu szeregowego

stopniowo przewijane w miarę odbierania nowych danych). Po usunięciu zaznaczenia tego pola wyboru użytkownik może ręcznie przewijać zawartość pola za pomocą pionowego paska przewijania.

Uruchamianie monitora portu szeregowego

Zanim będzie możliwe korzystanie z monitora portu szeregowego, należy ten monitor aktywować, umieszczając następujący wiersz w ciele funkcji `void setup()` szkicu:

```
Serial.begin(9600);
```

Wartość 9600 to tzw. *przepustowość*, czyli szybkość, z jaką dane będą przekazywane pomiędzy komputerem a płytką Arduino. Użyta wartość musi pasować do ustawień widocznych w prawym dolnym rogu okna monitora portu szeregowego (patrz rysunek 5.3).

Wysyłanie tekstu do monitora portu szeregowego

Aby wysłać do monitora portu szeregowego tekst, który zostanie wyświetlony w polu danych wyjściowych, wystarczy użyć funkcji `Serial.print()`:

```
Serial.print("Arduino dla każdego!");
```

Wywołanie funkcji powoduje wysłanie tekstu umieszczonego pomiędzy cudzysłowami do okna danych wyjściowych monitora portu szeregowego.

Alternatywnym rozwiązaniem jest użycie funkcji `Serial.println()`, która wyświetla tekst i umieszcza zaraz po nim znak nowego wiersza:

```
Serial.println("Arduino dla każdego!");
```

Wyświetlanie wartości zmiennych

W oknie monitora portu szeregowego można też wyświetlać wartości zmiennych. Na przykład poniższe wywołanie spowoduje wyświetlenie zawartości zmiennej `results`:

```
Serial.println(results);
```

Jeśli `results` jest zmienną typu `float`, wyrażenie w tej formie domyślnie wyświetli dwa miejsca po przecinku. Liczbę miejsc po przecinku (od zera do sześciu) można określić, przekazując na wejściu tej funkcji dodatkowy, drugi parametr (po nazwie zmiennej). Aby na przykład wyświetlić wartość zmiennej typu `float` z czterema miejscami po przecinku, należy użyć wywołania w postaci:

```
Serial.print(results,4);
```

Projekt nr 12: wyświetlanie temperatury w oknie monitora portu szeregowego

W tym projekcie zostaną wykorzystane urządzenia użyte wcześniej w projekcie nr 8. Tym razem temperatura w stopniach Celsjusza i Fahrenheita będzie wyświetlana w oknie monitora portu szeregowego. Szkic na potrzeby tego projektu będzie zawierał jedną funkcję określającą wartości temperatur i drugą wyświetlającą te wartości w oknie monitora portu szeregowego.

W środowisku IDE należy wpisać następujący kod:

```
// Projekt nr 12 — wyświetlanie temperatury w oknie monitora portu szeregowego

float celsius = 0;
float fahrenheit = 0;

void setup()
{
   Serial.begin(9600);
}

❶ void findTemps()
```

```
{
   float voltage = 0;
   float sensor = 0;
   // odczytuje wartość z czujnika temperatury i konwertuje ją na stopnie
   // Celsjusza i Fahrenheita
   sensor = analogRead(0);
   voltage = (sensor * 5000) / 1024;    // konwertuje wartość przekazaną przez
                                        // czujnik na miliwolty
   voltage = voltage - 500;             // odejmuje przesunięcie napięcia
   celsius = voltage / 10;              // konwertuje miliwolty na stopnie
                                        // Celsjusza
   fahrenheit = (1.8 * celsius) + 32;   // konwertuje stopnie Celsjusza na
                                        // stopnie Fahrenheita
}

❷ void displayTemps()
{
   Serial.print("Temperatura wynosi ");
   Serial.print(celsius, 2);
   Serial.print(" st. C / ");
   Serial.print(fahrenheit, 2);
   Serial.println(" st. F");
   // funkcja println() powoduje, że następny komunikat zostanie zapisany w nowym wierszu
}

void loop()
{
   findTemps();
   displayTemps();
   delay(1000);
}
```

Szkic realizuje wiele działań, ale jego struktura jest dość prosta dzięki utworzeniu dwóch funkcji: `findTemps()` w wierszu ❶ i `displayTemps()` w wierszu ❷. Obie funkcje są wywoływane w ciele funkcji `void loop()`, która dzięki tym wywołaniom jest bardzo prosta. Struktura tego szkicu dobrze ilustruje jedną z najważniejszych zalet funkcji — ułatwiają one zrozumienie kodu, poprawiają jego modułowość i umożliwiają wielokrotne wykorzystywanie gotowych elementów.

Po wysłaniu szkicu na płytkę Arduino należy odczekać kilka sekund, a następnie otworzyć okno monitora portu szeregowego. Pole tekstowe powinno zawierać komunikaty o temperaturze podobne do tych pokazanych na rysunku 5.4.

Diagnozowanie systemów za pomocą monitora portu szeregowego

Monitor portu szeregowego może z powodzeniem służyć do tzw. debugowania szkiców, czyli znajdowania i usuwania błędów w kodzie. Wystarczy umieścić w szkicu wyrażenia `Serial.println()` z krótkimi informacjami na temat aktualnie

Rysunek 5.4. Dane wynikowe generowane przez projekt nr 12

podejmowanych działań, aby następnie obserwować przebieg wykonywania szkicu przez system Arduino. W kodzie szkicu można na przykład umieścić następujący wiersz:

```
Serial.println("wykonywanie funkcji findTemps()");
```

Jeśli ten wiersz znajdzie się w ciele funkcji `findTemps()`, wyświetlane komunikaty pozwalają stwierdzić, kiedy system Arduino wykonuje wspomnianą funkcję.

Podejmowanie decyzji za pomocą wyrażeń while

Za pomocą wyrażeń `while()` można łatwo powtarzać instrukcje w ramach szkicu (instrukcje są powtarzane tak długo, jak długo użyty warunek jest spełniony). Warunek jest zawsze sprawdzany przed wykonaniem kodu zawartego w wyrażeniu `while()`. Na przykład konstrukcja `while ( temperature > 30 )` sprawdzi, czy wartość zmiennej `temperature` jest większa niż 30. Podczas tworzenia warunku w nawiasach okrągłych po słowie `while` można stosować dowolne operatory porównania.

Poniższy kod spowoduje, że system Arduino odliczy 10 sekund, a następnie wznowi wykonywanie programu:

```
int a = 0; // zmienna całkowitoliczbowa
while ( a < 10 )
{
      a = a + 1;
      delay(1000);
}
```

Szkic rozpoczyna się od przypisania zmiennej a wartości 0. Szkic zwiększa wartość tej zmiennej o 1, czeka sekundę (`delay(1000)`), a następnie powtarza ten proces aż do osiągnięcia wartości 10 przez zmienną a (`while ( a < 10 )`). W momencie osiągnięcia przez zmienną a wartości 10 porównanie w wyrażeniu `while` przestaje być prawdziwe, zatem system Arduino może kontynuować wykonywanie szkicu od miejsca następującego po zamykającym nawiasie klamrowym pętli `while`.

Konstrukcja do-while

W przeciwieństwie do wyrażenia `while` w konstrukcji `do-while()` test wyrażenia warunkowego jest wykonywany dopiero po wykonaniu kodu zawartego w tej strukturze. Oto przykład użycia tej konstrukcji:

```
int a = 0; // zmienna całkowitoliczbowa
do
{
      delay(1000);
      a = a + 1;
} while ( a < 100 );
```

Tym razem kod umieszczony pomiędzy nawiasami klamrowymi zostanie wykonany *przed* sprawdzeniem wyrażenia warunkowego (w tym przypadku `while ( a < 100 )`). Oznacza to, że nawet jeśli warunek nie jest spełniony, pętla przynajmniej raz zostanie wykonana. Decyzję o zastosowaniu konstrukcji `while` lub `do-while` należy podjąć zależnie od implementowanego rozwiązania.

Wysyłanie danych z monitora portu szeregowego do systemu Arduino

Aby wysłać dane z monitora portu szeregowego do systemu Arduino, należy włączyć w tym systemie tryb nasłuchiwania *bufora portu szeregowego*, czyli odbierania danych z zewnątrz za pośrednictwem pinów szeregowych (pinów cyfrowych nr 0 i 1), które są dodatkowo połączone z układem USB i przewodem podłączonym do komputera. Bufor portu szeregowego zawiera dane przychodzące, które użytkownik wpisuje w polu wejściowym w oknie monitora portu szeregowego.

Projekt nr 13: mnożenie liczby przez dwa

Aby zademonstrować proces wysyłania i odbierania danych za pośrednictwem monitora portu szeregowego, warto przeanalizować poniższy szkic. Szkic otrzymuje pojedynczą cyfrę wpisaną przez użytkownika, mnoży ją przez 2, a następnie wyświetla wynik w polu danych wynikowych monitora portu szeregowego.

```
// Projekt nr 13 — mnożenie liczby przez dwa

int number;

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   number = 0; // zmienna number jest gotowa na przypisanie nowej cyfry
   Serial.flush(); // usuwa ewentualne stare dane z bufora portu szeregowego
                   // przed przejściem w tryb nasłuchiwania
❶  while (Serial.available() == 0)
   {
      // nie podejmuje żadnych działań do momentu pojawienia się danych w buforze
      // portu szeregowego
   }
❷  while (Serial.available() > 0)
   {
      number = Serial.read() - '0';
      // odczytuje liczbę z bufora portu szeregowego,
      // usuwa przesunięcie ASCII dla zera: '0'
   }
   // Wyświetla odczytaną liczbę!
   Serial.print("Wpisana liczba: ");
   Serial.println(number);
   Serial.print(number);
   Serial.print(" pomnożone przez 2 równa się ");
   number = number * 2;
   Serial.println(number);
}
```

Funkcja `Serial.available()` w pierwszym wyrażeniu `while` ❶ zwraca wartość 0, jeśli użytkownik nie wpisał żadnych danych w monitorze portu szeregowego. Innymi słowy, pierwsza pętla `while` oznacza „nie podejmuj żadnych działań do momentu wpisania danych przez użytkownika". Drugie wyrażenie `while` ❷ wykrywa liczbę w buforze portu szeregowego i konwertuje otrzymany kod znaku (dane są wpisywane w formie tekstu) na właściwą wartość całkowito-

liczbową. Zaraz potem system Arduino wyświetla zarówno liczbę pobraną z bufora portu szeregowego, jak i iloczyn tej wartości i liczby 2.

Funkcja `Serial.flush()` na początku szkicu czyści bufor portu szeregowego na wypadek, gdyby w momencie uruchamiania programu zawierał jakieś nieoczekiwane dane — w ten sposób można przygotować bufor do odbierania właściwych danych. Okno monitora portu szeregowego po uruchomieniu tego szkicu pokazano na rysunku 5.5.

Rysunek 5.5. Przykładowe dane wejściowe i wyjściowe dla projektu nr 13

Mimo że użytkownik może już wpisywać w oknie monitora portu szeregowego dane liczbowe, które zostaną przetworzone przez system Arduino, stosowanie zmiennych typu `int` znacznie ogranicza zakres obsługiwanych wartości. Okazuje się jednak, że przedział wartości można znacznie zwiększyć, stosując zmienne typu `long`.

Zmienne typu long

Jeśli monitor portu szeregowego ma akceptować wartości dłuższe niż złożone z jednej cyfry, należy uzupełnić szkic o dodatkowy kod (pokazany w dalszej części tego rozdziału). Okazuje się jednak, że zmienne typu `int` mogą reprezentować wartości z dość ograniczonego przedziału — maksymalna wartość zmiennej tego typu wynosi 32 767. Na szczęście istnieje możliwość poszerzenia tego limitu dzięki zastosowaniu zmiennych typu `long`. Zmienna typu `long` może reprezentować liczby całkowite z przedziału od −2 147 483 648 do 2 147 483 647, czyli dużo większe wartości niż te obsługiwane przez zmienne typu `int` (od −32 768 do 32 767).

Projekt nr 14: stosowanie zmiennych typu long

W tym projekcie monitor portu szeregowego zostanie użyty do przekazywania zmiennych typu `long` i liczb złożonych z większej liczby cyfr. Szkic otrzymuje pewną liczbę cyfr wpisanych przez użytkownika, mnoży tę wartość przez 2, a następnie wyświetla wynik w polu danych wynikowych monitora portu szeregowego.

```
// Projekt nr 14 — stosowanie zmiennych typu long

long number = 0;
long a = 0;

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   number = 0; // zmienna number jest gotowa na przypisanie nowej cyfry
   Serial.flush(); // usuwa ewentualne stare dane z bufora portu szeregowego
                   // przed przejściem w tryb nasłuchiwania
   while (Serial.available() == 0)
   {
      // nie podejmuje żadnych działań do momentu wykrycia danych w buforze portu
      // szeregowego,
      // po pojawieniu się danych funkcja Serial.available() zwróci liczbę znaków
      // w buforze oczekujących na przetworzenie
   }
   // w buforze jest dostępny przynajmniej jeden znak — można przystąpić do obliczeń
   while (Serial.available() > 0)
   {
      // przenosi poprzednią cyfrę o jedną kolumnę w lewo;
      // innymi słowy, jeśli bufor zawiera dane, cyfra 1 jest zamieniana na liczbę 10
      number = number * 10;
      // odczytuje następną cyfrę z bufora i odejmuje znak 0,
      // aby przekonwertować wpisany znak na odpowiednią cyfrę
      a = Serial.read() - '0';
      // dodaje nowo odczytaną cyfrę do skumulowanej wartości liczbowej
      number = number + a;
      // krótkie opóźnienie umożliwia przesłanie ewentualnych dodatkowych danych
      // wykrywanych przez Serial.available()
      delay(5);
   }
   Serial.print("Wpisana liczba: ");
   Serial.println(number);
   Serial.print(number);
   Serial.print(" pomnożone przez 2 równa się ");
```

```
  number = number * 2;
  Serial.println(number);
}
```

W tym przykładzie zastosowano dwie pętle `while` umożliwiające systemowi Arduino odbieranie wielu cyfr z monitora portu szeregowego. Po wpisaniu pierwszej cyfry (skrajnie lewej cyfry składającej się na całą liczbę) szkic konwertuje otrzymany znak na liczbę, a następnie dodaje tę liczbę do zmiennej `number` reprezentującej łączną wartość. Jeśli liczba składa się tylko z jednej cyfry, szkic przechodzi do dalszych kroków. Jeśli użytkownik wpisze kolejną cyfrę (na przykład cyfrę 2 w ramach liczby 42), dotychczasowa wartość jest mnożona przez 10, aby przenieść pierwszą cyfrę o jedną pozycję w lewą stronę. Tak wyznaczona wartość jest następnie zwiększana o nową cyfrę. Cały cykl jest powtarzany do momentu dodania ostatniej, skrajnie prawej cyfry.

Dane wejściowe i wyjściowe tego szkicu pokazano na rysunku 5.6.

Rysunek 5.6. Przykładowe dane wejściowe i wyjściowe wyświetlone przez szkic dla projektu nr 14

Co dalej?

Mimo że projekty opisane w tym rozdziale na pierwszy rzut oka mogą się wydać mało efektowne, możliwość tworzenia własnych funkcji jest bardzo ważna dla dalszych projektów, ponieważ pozwala uprościć szkic i oszczędzić sporo czasu. Przekazana tutaj wiedza zostanie z powodzeniem wykorzystana w następnym rozdziale.

6

Liczby, zmienne i działania arytmetyczne

W tym rozdziale:

- nauczysz się generować liczby losowe;
- utworzysz elektroniczną kostkę do gry;
- poznasz liczby binarne;
- użyjesz układów scalonych rejestrów przesuwających, aby uzyskać więcej cyfrowych pinów wyjściowych;
- sprawdzisz swoją wiedzę o liczbach binarnych w formie quizu;
- poznasz tablice zmiennych;
- nauczysz się wyświetlać liczby za pomocą siedmiosegmentowych modułów LED;
- nauczysz się stosować funkcję matematyczną modulo;
- stworzysz cyfrowy termometr;
- opanujesz bitowe działania arytmetyczne;
- utworzysz stałe i ruchome obrazy na wyświetlaczach matrycowych LED.

W tym rozdziale wprowadzę wiele przydatnych mechanizmów, które otworzą nowe możliwości podczas realizacji projektów, w tym generator liczb losowych, nowe rodzaje funkcji matematycznych oraz techniki przechowywania zmiennych na listach i w tablicach. Co więcej, pokażę, jak używać wyświetlaczy LED (cyfrowych i matrycowych) do wyświetlania danych i prostych obrazów. I wreszcie podsumuję przekazaną wiedzę, tworząc między innymi grę i cyfrowy termometr.

Generowanie liczb losowych

Możliwość generowania liczb losowych w programie może być bardzo przydatna podczas tworzenia gier i efektów wizualnych. Liczby losowe można wykorzystać na przykład do zaimplementowania kostki do gry lub loterii w systemie Arduino, do utworzenia efektów świetlnych za pomocą diod LED lub do generowania efektów wizualnych i dźwiękowych na potrzeby quizu. Okazuje się jednak, że system Arduino nie oferuje możliwości generowania prawdziwych liczb losowych. Aby generowanie liczb losowych było możliwe, należy przekazać tzw. *wartość początkową* (ang. *seed*), czyli dowolną liczbę używaną jako punkt wyjścia dla generowanych wartości losowych.

Generowanie liczb losowych na podstawie napięcia na wolnym pinie

Najprostszym sposobem wygenerowania liczby losowej w systemie Arduino jest napisanie programu odczytującego napięcie na wolnym (niepodłączonym) pinie analogowym (na przykład na pinie analogowym nr 0) za pomocą następującego wiersza w ciele funkcji `void setup()`:

```
randomSeed(analogRead(0));
```

Mimo że do analogowego pinu wejściowego na płytce Arduino nie podłączono żadnych przewodów, zjawisko elektryczności statycznej otoczenia powoduje minimalne, mierzalne napięcie. Wysokość tego napięcia jest niemal losowa. Oznacza to, że odczyt napięcia na wolnym pinie można wykorzystać w roli wartości początkowej podczas generowania liczby losowej (samą liczbę można uzyskać za pomocą wywołania funkcji `random(lower, upper)`). Parametry `lower` i `upper` umożliwiają określenie odpowiednio minimalnej i maksymalnej wartości liczby losowej. Aby na przykład wygenerować liczbę z przedziału od 100 do 1000, należałoby użyć następującego kodu:

```
int a = 0;
a = random(100, 1001);
```

W powyższym kodzie zastosowano wartość maksymalną 1001 (zamiast 1000), ponieważ wyznaczany przedział jest prawostronnie otwarty (wartość maksymalna nie jest zaliczana do tego przedziału).

Aby wygenerować liczbę losową z przedziału od 0 do pewnej liczby, wystarczy przekazać tylko wartość maksymalną. Oto przykład generowania liczby losowej z przedziału od 0 do 6:

```
a = random(7);
```

Przykładowy szkic pokazany na listingu 6.1 wygeneruje liczby losowe z przedziału od 0 do 1000 oraz z przedziału od 10 do 50.

Listing 6.1. Generator liczb losowych

```
// Listing 6.1
int r = 0;

void setup()
{
   randomSeed(analogRead(0));
   Serial.begin(9600);
}

void loop()
{
   Serial.print("Liczba losowa z przedziału od 0 do 1000: ");
   r = random(0, 1001);
   Serial.println(r);
   Serial.print("Liczba losowa z przedziału od 10 do 50: ");
   r = random(10, 51);
   Serial.println(r);
   delay(1000);
}
```

Wynik wykonania tego szkicu wyświetlony w monitorze portu szeregowego pokazano na rysunku 6.1.

Skoro wyjaśniłem już, jak generować liczby losowe, czas wykorzystać tę wiedzę do utworzenia elektronicznej kostki do gry.

Rysunek 6.1. Dane wynikowe szkicu z listingu 6.1

Projekt nr 15: tworzenie elektronicznej kostki do gry

Celem tego projektu jest włączenie jednej z sześciu diod LED (wybranej losowo), czyli naśladowanie rzutu kostką do gry. Program wybierze losową liczbę z przedziału od 1 do 6, a następnie włączy odpowiednią diodę LED, aby zaprezentować wynik. Na potrzeby tego szkicu zostanie utworzona funkcja losowo wybierająca jedną z sześciu diod LED podłączonych do płytki Arduino i włączająca tę diodę na określony czas. Zaraz po włączeniu lub ponownym uruchomieniu płytki Arduino działającej pod kontrolą tego szkicu na pewien czas zostanie włączona losowo wybrana dioda LED; z czasem tempo włączania kolejnych diod LED będzie malało. Dioda LED włączona na podstawie ostatniej liczby losowej będzie się świeciła do momentu wyłączenia lub ponownego uruchomienia płytki Arduino.

Sprzęt

Budowa elektronicznej kostki do gry wymaga użycia następujących urządzeń:

- sześciu diod LED dowolnego koloru (od LED1 do LED6);
- jednego rezystora 560 Ω;
- różnych przewodów;
- płytki uniwersalnej średniej wielkości;
- płytki Arduino i przewodu USB.

Schemat

Ponieważ jednocześnie będzie włączona tylko jedna dioda LED, wystarczy zastosować jeden rezystor ograniczający natężenie pomiędzy katodami diod LED a pinem GND. Schemat obwodu elektronicznej kostki do gry pokazano na rysunku 6.2.

Szkic

Oto szkic dla projektu elektronicznej kostki do gry:

```
// Projekt nr 15 — tworzenie elektronicznej kostki do gry
void setup()
{
   randomSeed(analogRead(0));    // wartość początkowa generatora liczb losowych
   for ( int z = 1 ; z < 7 ; z++ ) // piny od 1. do 6. będą służyły do wysyłania
                                   // sygnałów wyjściowych (do diod LED)
   {
      pinMode(z, OUTPUT);
   }
}
```

Rysunek 6.2. Schemat obwodu dla projektu nr 15

```
void randomLED(int del)
{
   int r;
   r = random(1, 7);        // losuje liczbę z przedziału od 1 do 6
   digitalWrite(r, HIGH);   // włącza jedną z diod LED podłączonych
                            // do pinów od 1. do 6.
   if (del > 0)
   {
❶   delay(del);             // utrzymuje włączoną diodę LED przez określony czas
   }
❷  else if (del == 0)
   {
      do                    // wartość opóźnienia jest równa zero, zatem dioda
                            // LED pozostanie włączona na stałe
      {}
```

```
❸     while (1);
   }
   digitalWrite(r, LOW); // wyłącza diodę LED
}

void loop()
{
   int a;
   // losuje i włącza kolejne diody LED dla ciekawszego efektu
   for ( a = 0 ; a < 100 ; a++ )
   {
      randomLED(50);
   }
   // kolejne diody są losowane i włączane coraz wolniej
❹  for ( a = 1 ; a <= 10 ; a++ )
   {
      randomLED(a * 100);
   }
   // szkic kończy działanie, włączając diodę LED wylosowaną w ostatnim cyklu
   randomLED(0);
}
```

Szkic aktywuje cyfrowe piny wyjściowe w pętli umieszczonej w ciele funkcji `void setup()`. Funkcja `randomLED()` otrzymuje na wejściu liczbę całkowitą, która jest następnie stosowana w wywołaniu funkcji `delay()`❶ i — tym samym — określa czas, przez jaki dioda LED jest włączona. Jeśli na wejściu funkcji `randomLED()` została przekazana wartość 0 ❷, dioda LED jest włączana na stałe, ponieważ w wierszu ❸ zastosowano następującą konstrukcję:

```
do {} while (1);
```

Ten zapis powoduje wykonywanie pętli w nieskończoność, ponieważ 1 zawsze jest równe 1.

Aby symulować rzut kostką, należy ponownie uruchomić system Arduino i powyższy szkic. Początkowo losowa dioda LED jest włączana 100 razy na 50 milisekund, ale z czasem (do momentu włączenia ostatniej diody) zmiany świecących diod LED następują coraz wolniej ❹. Po pierwszych krótkich mignięciach cykl włączania diod LED jest spowalniany — diody są włączane 10 razy na czas od 100 do 1000 milisekund. Wydłużenie czasu świecenia diod LED ma na celu symulowanie coraz wolniejszego ruchu rzuconej kostki aż do osiągnięcia właściwej wartości — właśnie dlatego system Arduino na stałe włącza diodę LED wylosowaną jako ostatnia:

```
randomLED(0);
```

Modyfikowanie szkicu

Projekt można zmodyfikować na wiele różnych sposobów. System można uzupełnić na przykład o dodatkowe sześć diod LED, aby symulować rzucanie dwiema kostkami do gry. Warto też rozważyć wykorzystanie wbudowanej diody LED, tak aby migała tyle razy, ile wyniesie wylosowana liczba. Zachęcam do wykorzystania wyobraźni i nowych umiejętności podczas samodzielnych eksperymentów!

Krótkie wprowadzenie w świat liczb binarnych

O ile większość dzieci uczy się liczyć w dziesiętnym systemie liczbowym, o tyle komputery (w tym platforma Arduino) operują na liczbach systemu dwójkowego (binarnego). Liczby binarne składają się wyłącznie z zer i jedynek, na przykład 10101010. W systemie binarnym każda cyfra od prawej do lewej strony reprezentuje potęgę liczby 2 (wykładnik potęgi zależy od pozycji cyfry i zwiększa się od prawej do lewej strony). Aby określić łączną wartość liczby binarnej, należy zsumować wspomniane potęgi liczby 2.

Warto przeanalizować na przykład liczbę binarną 11111111 (patrz tabela 6.1). Aby przekonwertować liczbę 11111111 na system dziesiętny, należy zsumować wszystkie potęgi (zgodnie ze schematem pokazanym w dolnym wierszu tabeli):

128 + 64 + 32 + 16 + 8 + 4 + 2 + 1

Tabela 6.1. Przykład konwersji liczby binarnej na liczbę dziesiętną

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	Liczba binarna
128	64	32	16	8	4	2	1	Liczba dziesiętna

Liczba binarna 11111111 odpowiada wartości 255 w systemie dziesiętnym. Liczba binarna złożona z ośmiu cyfr (bitów) reprezentuje 1 bajt danych — 1 bajt może zawierać wartość liczbową z przedziału od 0 do 255. Skrajnie lewy bit określa się mianem *najbardziej znaczącego bitu* (ang. *most significant bit — MSB*), a skrajnie prawy bit jest nazywany *najmniej znaczącym bitem* (ang. *least significant bit — LSB*).

Liczby binarne doskonale sprawdzają się jako reprezentacja pewnych typów danych, na przykład stanu diod LED (włączonych lub wyłączonych), prawdy i fałszu oraz stanu cyfrowych pinów wyjściowych. Liczby binarne składają się na wszystkie typy danych stosowane w komputerach.

Zmienne typu byte

Jednym ze sposobów przechowywania liczb binarnych jest stosowanie *zmiennych typu `byte`*. Zmienną typu `byte` można utworzyć na przykład za pomocą następującego kodu:

```
byte outputs = B11111111;
```

Litera B przed przypisywaną liczbą wymusza na systemie Arduino traktowanie tej wartości jako liczby binarnej (w tym przypadku 11111111, czyli odpowiednika wartości 255 w systemie dziesiętnym) zamiast jako liczby dziesiętnej. Inny przykład pokazano na listingu 6.2.

Listing 6.2. Przykład stosowania liczb binarnych

```
// Listing 6.2

byte a;

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   for ( int count = 0 ; count < 256 ; count++ )
   {
      a = count;
      Serial.print("Liczba dziesiętna: ");
   ❶ Serial.print(a, DEC);
      Serial.print(" Liczba binarna: ");
   ❷ Serial.println(a, BIN);
      delay(1000);
   }
}
```

Wartość zmiennej typu `byte` jest wyświetlana zarówno w systemie dziesiętnym ❶, jak i w systemie dwójkowym ❷ — wystarczy za pośrednictwem drugiego parametru funkcji `Serial.print()` przekazać odpowiednio wartość `DEC` i `BIN`. Po wysłaniu tego szkicu na płytkę Arduino okno monitora portu szeregowego powinno zawierać dane podobne do tych z rysunku 6.3.

Zwiększanie liczby dostępnych pinów cyfrowych za pomocą rejestrów przesuwających

Płytka Arduino oferuje 13 pinów cyfrowych, które mogą być używane do przekazywania sygnałów wyjściowych. W pewnych przypadkach liczba dostępnych pinów okazuje się jednak za mała. Aby zwiększyć liczbę pinów wyjściowych, wystarczy użyć *rejestru przesuwającego* (ang. *shift register*). Rejestr przesuwający

Rysunek 6.3. Dane wynikowe szkicu z listingu 6.2

to układ scalony z ośmioma cyfrowymi pinami wyjściowymi. Rejestrem można sterować, wysyłając bajt danych. W projektach opisywanych w tej książce będzie stosowany rejestr przesuwający 74HC595 (patrz rysunek 6.4).

Rysunek 6.4. Układ rejestru przesuwającego 74HC595

Rejestr przesuwający 74HC595 udostępnia osiem cyfrowych pinów wyjściowych, które mogą być wykorzystywane podobnie do cyfrowych pinów wyjściowych na samej płytce Arduino. Rejestr przesuwający zajmuje trzy cyfrowe piny wyjściowe na płytce Arduino, zatem zastosowanie tego urządzenia pozwala zwiększyć łączną liczbę dostępnych pinów o pięć.

Zasada działania rejestru przesuwającego jest bardzo prosta — należy wysłać do rejestru bajt danych (8 bitów), na podstawie którego rejestr włączy lub wyłączy każdy ze swoich ośmiu pinów wyjściowych. Bity składające się na wysłany bajt danych są przypisywane do pinów wynikowych w kolejności od największego do najmniejszego. Oznacza to, że skrajnie lewy bit danych reprezentuje stan pinu nr 7 rejestru przesuwającego, a skrajnie prawy bit reprezentuje stan pinu wyjściowego nr 0. Jeśli na przykład do rejestru przesuwającego zostanie wysłana wartość B10000110, zostaną włączone piny wyjściowe nr 7, 2 i 1 oraz wyłączone piny nr 0, 3, 4, 5 i 6. Ten stan będzie utrzymany do momentu wysłania następnego bajta danych lub wyłączenia zasilania.

Istnieje możliwość łączenia wielu rejestrów przesuwających, dzięki czemu można zyskać jeszcze więcej dodatkowych pinów wyjściowych, korzystając z tych samych trzech pinów na płytce Arduino — oznacza to, że rejestry przesuwające

są bardzo wygodne, jeśli płytka Arduino ma sterować na przykład dużą liczbą diod LED. Spróbujmy teraz zbudować na bazie rejestru przesuwającego system obsługujący wyświetlacz liczb binarnych.

Projekt nr 16: tworzenie wyświetlacza liczb binarnych złożonego z diod LED

W tym projekcie zostanie użytych osiem diod LED do wyświetlania liczb binarnych z przedziału od 0 do 255. W szkicu zostanie wykorzystana pętla `for`, która odliczy od 0 do 255 i wyśle każdą z wartości do rejestru przesuwającego — na podstawie otrzymywanych bajtów rejestr włączy diody LED reprezentujące kolejne wartości.

Sprzęt

Realizacja tego projektu wymaga następującego sprzętu:

- rejestru przesuwającego 74HC595;
- ośmiu diod LED (od LED1 do LED8);
- ośmiu rezystorów 560 Ω (od R1 do R8);
- płytki uniwersalnej;
- różnych przewodów;
- płytki Arduino i przewodu USB.

Łączenie rejestru przesuwającego 74HC595

Symbol rejestru 74HC595 stosowany na schematach obwodów pokazano na rysunku 6.5.

Rysunek 6.5. Symbol rejestru przesuwającego 74HC595

Rejestr przesuwający udostępnia 16 pinów:

- Pin nr 15 oraz piny od 1. do 7. (czyli łącznie osiem pinów) służą do przekazywania sygnałów wyjściowych — piny wyjściowe oznaczono odpowiednio od Q0 do Q7.
- Pin Q7 służy do wysyłania pierwszego bitu otrzymanego przez rejestr; pin Q0 wysyła ostatni otrzymany bit.
- Pin nr 8 należy połączyć z uziemieniem (GND).
- Pin nr 9 służy do wysyłania danych do ewentualnego innego rejestru przesuwającego.
- Pin nr 10 zawsze jest połączony ze źródłem napięcia 5 V (na przykład z pinem 5 V na płytce Arduino).
- Piny nr 11 i 12 to odpowiednio *zegar* (ang. *clock*) i *zatrzask* (ang. *latch*).
- Pin nr 13 (*OE*, od ang. *output enable*) zwykle jest łączony z uziemieniem.
- Pin nr 14 służy do odbierania danych bitowych wysyłanych przez płytkę Arduino.
- Pin nr 16 służy do zasilania prądem 5 V z płytki Arduino.

Aby lepiej zrozumieć rozkład pinów, pomiędzy pinami nr 1 i 16 jest widoczne półokrągłe wycięcie na powierzchni rejestru (patrz rysunek 6.4).

Same piny ponumerowano kolejno odwrotnie do kierunku ruchu wskazówek zegara (patrz rysunek 6.6, na którym pokazano kompletny schemat obwodu wyświetlacza liczb binarnych).

Rysunek 6.6. Schemat obwodu dla projektu nr 16

UWAGA *Po przygotowaniu obwodu dla tego przykładu nie należy rozłączać elementów. Obwód będzie potrzebny także podczas realizacji kolejnych projektów.*

Szkic

Czas na analizę szkicu dla tego projektu:

```
// Projekt nr 16 — tworzenie wyświetlacza liczb binarnych złożonego z diod LED
#define DATA 6    // pin cyfrowy nr 6 połączony z pinem nr 14 układu 74HC595
#define LATCH 8   // pin cyfrowy nr 8 połączony z pinem nr 12 układu 74HC595
#define CLOCK 10  // pin cyfrowy nr 10 połączony z pinem nr 11 układu 74HC595
void setup()
{
   pinMode(LATCH, OUTPUT);
   pinMode(CLOCK, OUTPUT);
   pinMode(DATA, OUTPUT);
}
void loop()
{
   int i;
   for ( i = 0; i < 256; i++ )
   {
      digitalWrite(LATCH, LOW);
      shiftOut(DATA, CLOCK, MSBFIRST, i);
      digitalWrite(LATCH, HIGH);
      delay(200);
   }
}
```

W funkcji `void setup()` tego szkicu ustawiono trzy piny połączone z rejestrem przesuwającym jako piny wyjściowe, a następnie zdefiniowano w ciele funkcji `void loop()` pętlę odliczającą od 0 do 255. Właśnie ta pętla jest sercem programu. Podczas wysyłania bajta danych (na przykład wartości 240, czyli B11110000) do rejestru przesuwającego w pętli `for` są wykonywane trzy operacje:

- Szkic ustawia stan `LOW` na pinie nr 12 — oznacza to, że płytka Arduino wyśle na ten pin sygnał niski za pośrednictwem swojego pinu nr 8. Ten krok jest tylko przygotowaniem do właściwego przekazania danych i ponownego ustawienia stanu `HIGH` na pinie nr 12 po zakończeniu działania funkcji `shiftOut()`.
- Szkic wysyła bajt danych (na przykład wartość B11110000) za pośrednictwem pinu cyfrowego nr 6 na płytce Arduino do rejestru przesuwającego i jednocześnie wskazuje funkcji `shiftOut()` właściwy sposób interpretacji tego bajta. W przypadku zastosowania trybu `LSBFIRST` zostałyby włączone diody LED od 1 do 4 (pozostałe diody byłyby wyłączone). W kodzie zastosowano tryb `MSBFIRST`, zatem dla wspomnianej wartości zostaną włączone diody LED od 5 do 8 (pozostałe będą wyłączone).

- I wreszcie szkic ustawia stan `HIGH` (5 V) na pinie nr 12. Stan wysoki jest dla rejestru przesuwającego sygnałem, że wszystkie bity zostały przekazane. Od tego momentu rejestr przekazuje swoje dane wyjściowe na podstawie otrzymanego bajta.

Projekt nr 17: implementacja binarnego quizu

Podczas przygotowywania systemu quizu w tym projekcie zostaną wykorzystane liczby losowe, monitor portu szeregowego oraz obwód przygotowany z myślą o projekcie nr 16. System Arduino będzie wyświetlał losową liczbę binarną za pomocą diod LED, a zadaniem użytkownika będzie wpisanie tej liczby (w wersji dziesiętnej) za pośrednictwem monitora portu szeregowego. Monitor portu szeregowego zostanie użyty także do wyświetlenia komunikatu na temat poprawności udzielonej odpowiedzi. Po wyświetleniu tego komunikatu gra będzie kontynuowana — zostanie wyświetlona nowa liczba.

Algorytm

Algorytm tego projektu można podzielić na trzy funkcje. Funkcja `displayNumber()` odpowiada za wyświetlenie liczby binarnej za pomocą diod LED. Funkcja `get ↪Answer()` otrzymuje na wejściu liczbę wpisaną w oknie monitora portu szeregowego i wyświetla ją w polu danych wyjściowych. I wreszcie funkcja `checkAnswer()` porównuje liczbę wpisaną przez użytkownika z wygenerowaną liczbą losową i wyświetla komunikat z informacją o poprawnej lub błędnej odpowiedzi.

Szkic

Szkic dla tego projektu generuje liczbę losową z przedziału od 0 do 255, wyświetla ją w formie binarnej (za pomocą diod LED), prosi użytkownika o podanie odpowiedzi, a następnie wyświetla wynik w oknie monitora portu szeregowego. Wszystkie funkcje użyte w tym szkicu były już omawiane, zatem mimo dość długiego kodu zrozumienie tego programu nie powinno sprawić najmniejszego kłopotu. Aby mieć pewność, że szkic będzie prawidłowo interpretowany, pewne aspekty wyjaśnię w komentarzach i w tekście poniżej.

```
// Projekt nr 17 — implementacja binarnego quizu

#define DATA 6              // połączenie z pinem nr 14 rejestru 74HC595
#define LATCH 8             // połączenie z pinem nr 12 rejestru 74HC595
#define CLOCK 10            // połączenie z pinem nr 11 rejestru 74HC595

int number = 0;
int answer = 0;

❶ void setup()
```

```
{
   pinMode(LATCH, OUTPUT);     // ustawia piny rejestru 74HC595
   pinMode(CLOCK, OUTPUT);
   pinMode(DATA, OUTPUT);
   Serial.begin(9600);
   randomSeed(analogRead(0)); // inicjalizuje generator liczb losowych
   displayNumber(0);          // wyłącza diody LED
}

❷ void displayNumber(byte a)
{
   // wysyła bajt w celu wyświetlenia za pomocą diod LED
   digitalWrite(LATCH, LOW);
   shiftOut(DATA, CLOCK, MSBFIRST, a);
   digitalWrite(LATCH, HIGH);
}

❸ void getAnswer()
{
   // pobiera odpowiedź od gracza
   int z = 0;
   Serial.flush();
   while (Serial.available() == 0)
   {
      // nie podejmuje żadnych działań do momentu wykrycia danych w buforze portu
      // szeregowego
   }
   // w buforze jest dostępny przynajmniej jeden znak — można przystąpić do obliczeń
   while (Serial.available() > 0)
   {
      // przenosi poprzednią cyfrę o jedną kolumnę w lewo;
      // innymi słowy, jeśli bufor zawiera dane, cyfra 1 jest zamieniana na liczbę 10
      answer = answer * 10;
      // odczytuje następną cyfrę z bufora i odejmuje znak '0',
      // aby przekonwertować wpisany znak na odpowiednią cyfrę
      z = Serial.read() - '0';
      // dodaje nowo odczytaną cyfrę do skumulowanej wartości liczbowej
      answer = answer + z;
      // krótkie opóźnienie umożliwia przesłanie ewentualnych dodatkowych danych
      // wykrywanych przez Serial.available()
      delay(5);
   }
   Serial.print("Wpisana liczba: ");
   Serial.println(answer);
}

❹ void checkAnswer()
{
   // sprawdza odpowiedź gracza i wyświetla wynik
   if (answer == number) // prawidłowa odpowiedź!
```

```
   {
      Serial.print("Odpowiedź prawidłowa! ");
      Serial.print(answer, BIN);
      Serial.print(" to ");
      Serial.println(number);
      Serial.println();
   }
   else                    // błędna odpowiedź
   {
      Serial.print("Błędna odpowiedź! ");
      Serial.print(number, BIN);
      Serial.print(" to ");
      Serial.println(number);
      Serial.println();
   }
   answer = 0;
   delay(10000); // gracz ma trochę czasu na analizę swojej odpowiedzi
}

void loop()
{
   number = random(256);
   displayNumber(number);
   Serial.println("Ile wyniesie ta liczba binarna w systemie dziesiętnym? ");
   getAnswer();
   checkAnswer();
}
```

Przeanalizujmy teraz działanie tego szkicu. Funkcja `void setup()` ❶ konfiguruje cyfrowe piny wyjściowe pod kątem wysyłania sygnału do rejestru przesuwającego, uruchamia monitor portu szeregowego i określa wartość początkową dla generatora liczb losowych. Funkcja niestandardowa `displayNumber()` ❷ otrzymuje na wejściu bajt danych i wysyła go do rejestru przesuwającego, który z kolei wyświetla binarną reprezentację tego bajta za pomocą diod LED (tak jak w projekcie nr 16). Funkcja niestandardowa `getAnswer()`❸ otrzymuje liczbę wpisaną przez użytkownika w oknie monitora portu szeregowego (tak jak w projekcie nr 14) i wyświetla tę liczbę (patrz rysunek 6.7).

Funkcja `checkAnswer()`❹ porównuje liczbę wpisaną przez użytkownika (uzyskaną przez funkcję `getAnswer()`) z liczbą losową wygenerowaną przez szkic w ciele funkcji `void loop()`. Gracz jest następnie informowany o podaniu prawidłowej lub błędnej odpowiedzi. Szkic dodatkowo wyświetla obie wartości, aby gracz mógł przeanalizować ewentualne błędy. I wreszcie w głównej funkcji `void loop()`, która steruje wykonywaniem całego programu, system Arduino generuje liczbę losową na potrzeby quizu, a następnie wywołuje funkcję wyświetlającą tę wartość za pomocą diod LED, funkcję pobierającą odpowiedź użytkownika i funkcję sprawdzającą poprawność tej odpowiedzi.

Przebieg gry wyświetlony w oknie monitora portu szeregowego pokazano na rysunku 6.7.

Rysunek 6.7. Projekt nr 17 w akcji

Tablice

Tablica (ang. *array*) to zbiór zmiennych lub wartości pogrupowanych w sposób umożliwiający odwoływanie się do tych danych jako do jednej całości. Tablice są bardzo przydatne podczas przetwarzania dużej ilości powiązanych danych — znacznie ułatwiają ich organizowanie.

Definiowanie tablicy

Każda wartość w tablicy jest określana mianem jej *elementu*. Przypuśćmy na przykład, że dysponujemy sześcioma zmiennymi typu `float` reprezentującymi temperatury z ostatnich sześciu godzin; zamiast nadawać każdej z tych zmiennych odrębną nazwę, można zdefiniować sześcioelementową tablicę nazwaną `temperatures`:

```
float temperatures[6];
```

Podczas definiowania tablicy można też określić wartości jej elementów. W takim przypadku nie jest konieczne definiowanie wielkości tablicy. Oto przykład:

```
float temperatures[]={11.1, 12.2, 13.3, 14.4, 15.5, 16.6};
```

Łatwo zauważyć, że tym razem nie określono wprost wielkości tablicy w nawiasach kwadratowych (`[]`) — rozmiar tablicy jest określany na podstawie liczby elementów zdefiniowanych w nawiasach klamrowych (`{}`).

Odwoływanie się do wartości w tablicy

Elementy tablicy są numerowane od lewej do prawej strony; pierwszy element jest oznaczany indeksem 0, zatem tablica `temperatures[]` zawiera elementy od 0 do 5. Odwołania do poszczególnych wartości w ramach tablicy wymagają tylko umieszczania ich numerów w nawiasach kwadratowych. Aby na przykład zmienić wartość pierwszego elementu tablicy `temperatures[]` (który obecnie ma wartość `11.1`) na `12.34`, należy użyć wyrażenia w tej formie:

```
temperatures[0] = 12.34;
```

Zapisywanie i odczytywanie danych przechowywanych w tablicach

Kod na listingu 6.3 demonstruje sposoby zapisywania i odczytywania wartości przechowywanych w pięcioelementowej tablicy. Pierwsza pętla `for` tego szkicu zapisuje liczbę losową w każdym elemencie tablicy; druga pętla odczytuje elementy tablicy i wyświetla je w oknie monitora portu szeregowego.

Listing 6.3. Przykład odczytywania i zapisywania danych zawartych w tablicy

```
// Listing 6.3
void setup()
{
   Serial.begin(9600);
   randomSeed(analogRead(0));
}

int array[5];    // definiuje tablicę złożoną z pięciu elementów całkowitoliczbowych

void loop()
{
   int i;
   Serial.println();
   for ( i = 0 ; i < 5 ; i++ )    // zapisuje wartości w tablicy
   {
      array[i] = random(10);       // liczby losowe z przedziału od 0 do 9
   }
   for ( i = 0 ; i < 5 ; i++ )    // wyświetla zawartość tablicy
   {
      Serial.print("array[");
      Serial.print(i);
      Serial.print("] ma wartość ");
      Serial.println(array[i]);
   }
   delay(5000);
}
```

Dane wynikowe tego szkicu wyświetlone w oknie monitora portu szeregowego pokazano na rysunku 6.8.

Rysunek 6.8. Dane wynikowe szkicu z listingu 6.3

Skoro wyjaśniłem już, jak korzystać z liczb binarnych, rejestrów przesuwających i tablic, czas wykorzystać tę wiedzę w praktyce. W następnym podrozdziale pokażę, jak stosować wyświetlacze cyfrowe.

Siedmiosegmentowe wyświetlacze LED

Diody LED są bardzo przydatne, ale zakres danych, które można wyświetlać za pomocą pojedynczych świateł, jest bardzo ograniczony. W tym podrozdziale cyfry będą po raz pierwszy wyświetlane za pomocą siedmiosegmentowych wyświetlaczy LED (przykłady takich wyświetlaczy pokazano na rysunku 6.9).

Rysunek 6.9. Moduły siedmiosegmentowych wyświetlaczy LED

Wyświetlacze tego typu doskonale nadają się do wyświetlania cyfr, dlatego są powszechnie stosowane w cyfrowych budzikach, szybkościomierzach i innych urządzeniach wyświetlających wartości liczbowe. Każdy moduł siedmiosegmentowego wyświetlacza LED składa się z ośmiu diod LED. Moduły są dostępne w różnych kolorach. Aby ograniczyć liczbę pinów używanych przez wyświetlacz, anody lub katody wszystkich diod LED są połączone w ramach tzw. *wspólnej anody* lub *wspólnej katody* wyświetlacza. We wszystkich projektach opisywanych w tej książce będą stosowane moduły ze wspólną katodą.

Diody LED wyświetlacza oznaczono literami od *A* do *G* oraz parą liter *DP* (od ang. *decimal point*). Dla każdego segmentu (każdej diody LED) istnieje odrębny pin anody, natomiast katody wszystkich diod są podłączone do jednego pinu wspólnej katody. Układ siedmiosegmentowych wyświetlaczy LED zawsze jest zgodny ze schematem pokazanym na rysunku 6.10 — segment A znajduje się na samej górze, segment B znajduje się na prawo od niego itd. Oznacza to, że aby wyświetlić na przykład cyfrę 7, należy włączyć zasilanie segmentów A, B i C.

Rysunek 6.10. Rozkład diod LED typowego modułu wyświetlacza siedmiosegmentowego

Układ pinów w różnych modułach wyświetlaczy LED może być inny (zależnie od producenta), jednak podstawowy układ segmentów zawsze jest taki jak na rysunku 6.10. Przed użyciem modułu wyświetlacza zawsze należy sprawdzić dokumentację producenta — w ten sposób można oszczędzić sporo czasu podczas identyfikacji znaczenia poszczególnych pinów.

Na schematach obwodów będzie stosowany symbol modułów wyświetlaczy siedmiosegmentowych LED pokazany na rysunku 6.11.

Rysunek 6.11. Symbol modułu wyświetlacza siedmiosegmentowego

Sterowanie wyświetlaczem LED

Do sterowania wyświetlaczem LED zostanie użyta metoda opisana w projekcie nr 17 — piny od A do DP zostaną podłączone do pinów wyjściowych od Q0 do Q7 rejestru przesuwającego. Tabela 6.2 zawiera schemat ułatwiający identyfikację segmentów, które należy włączyć i wyłączyć, aby wyświetlić określoną cyfrę lub literę.

Tabela 6.2. *Schemat wartości wysyłanych do wyświetlacza siedmiosegmentowego*

Rejestr przesuwający	**Q0**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**	**Q6**	**Q7**	
Segment	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**DP**	**Liczba dziesiętna**
0	1	1	1	1	1	1	0	0	252
1	0	1	1	0	0	0	0	0	96
2	1	1	0	1	1	0	1	0	218
3	1	1	1	1	0	0	1	0	242
4	0	1	1	0	0	1	1	0	102
5	1	0	1	1	0	1	1	0	182
6	1	0	1	1	1	1	1	0	190
7	1	1	1	0	0	0	0	0	224
8	1	1	1	1	1	1	1	0	254
9	1	1	1	1	0	1	1	0	246
A	1	1	1	0	1	1	1	0	238
B	0	0	1	1	1	1	1	0	62
C	1	0	0	1	1	1	0	0	156
D	0	1	1	1	1	0	1	0	122
E	1	0	0	1	1	1	1	0	158
F	1	0	0	0	1	1	1	0	142

Pierwszy wiersz tabeli zawiera pin wyjściowy rejestru przesuwającego, który steruje segmentami reprezentowanymi w drugim wierszu. Dalsze wiersze zawierają cyfry i litery wraz z odpowiednimi wartościami binarnymi i dziesiętnymi, które należy wysłać do rejestru przesuwającego.

Aby wyświetlić na przykład cyfrę 7 (patrz rysunek 6.12), należy włączyć diody LED w segmentach A, B i C, czyli włączyć zasilanie na pinach wyjściowych rejestru przesuwającego oznaczonych etykietami Q0, Q1 i Q2. Oznacza to, że do rejestru przesuwającego należy wysłać bajt B1110000 (z ustawionym trybem `LSBFIRST` w wywołaniu funkcji `shiftOut()`), aby włączyć zasilanie na trzech pierwszych pinach wyjściowych i — tym samym — włączyć odpowiednie diody LED wyświetlacza.

Rysunek 6.12. Cyfra 7 na wyświetlaczu LED

W następnym przykładzie zbudujemy obwód wyświetlający kolejno cyfry od 0 do 9 oraz litery od A do F. Cały cykl zostanie powtórzony z włączoną diodą LED punktu dziesiętnego (DP).

Projekt nr 18: tworzenie wyświetlacza jednocyfrowego

W tym projekcie zbudujemy obwód sterujący wyświetlaczem jednocyfrowym.

Sprzęt

Do realizacji tego projektu będzie potrzebny następujący sprzęt:

- rejestr przesuwający 74HC595;
- siedmiosegmentowy wyświetlacz LED ze wspólną katodą;
- osiem rezystorów 560 Ω (od R1 do R8);
- duża płytka uniwersalna;
- różne przewody;
- płytka Arduino i przewód USB.

Schemat

Schemat obwodu dla tego projektu pokazano na rysunku 6.13.

Podczas łączenia modułu wyświetlacza LED z rejestrem przesuwającym piny wyświetlacza od A do G należy połączyć z pinami rejestru odpowiednio od Q0 do Q6, a pin DP należy połączyć z pinem Q7.

Szkic

W szkicu dla projektu nr 18 wartości dziesiętne (patrz tabela 6.2) są przechowywane w tablicy `int digits[]`. W ciele funkcji `void loop()` wartości są kolejno wysyłane do rejestru przesuwającego ❶, a następnie cały ten proces jest powtarzany z włączoną diodą punktu dziesiętnego (wystarczy wysyłać do rejestru przesuwającego wartość powiększoną o 1 ❷):

Rysunek 6.13. Schemat obwodu dla projektu nr 18

```
// Projekt nr 18 — tworzenie wyświetlacza jednocyfrowego
#define DATA 6          // połączenie z pinem nr 14 rejestru 74HC595
#define LATCH 8         // połączenie z pinem nr 12 rejestru 74HC595
#define CLOCK 10        // połączenie z pinem nr 11 rejestru 74HC595

// definiuje tablicę z dziesiętną reprezentacją segmentów wyświetlacza dla cyfr od 0 do 9
// i liter od A do F (patrz tabela 6.2)
int digits[] = {252, 96, 218, 242, 102, 182, 190, 224, 254, 246, 238, 62,
156, 122, 158, 142};

void setup()
{
   pinMode(LATCH, OUTPUT);
   pinMode(CLOCK, OUTPUT);
   pinMode(DATA, OUTPUT);
}

void loop()
{
   int i;
   for ( i = 0 ; i < 16 ; i++ ) // wyświetla cyfry 0 – 9 i litery A – F
   {
      digitalWrite(LATCH, LOW);
   ❶ shiftOut(DATA, CLOCK, LSBFIRST, digits[i]);
      digitalWrite(LATCH, HIGH);
      delay(250);
   }
```

```
  for ( i = 0 ; i < 16 ; i++ ) // wyświetla cyfry 0 – 9 i litery A – F z włączoną
                               // diodą DP
  {
     digitalWrite(LATCH, LOW);
❷    shiftOut(DATA, CLOCK, LSBFIRST, digits[i]+1);
❸    // +1 powoduje włączenie bitu dla diody DP
     digitalWrite(LATCH, HIGH);
     delay(250);
  }
}
```

Siedmiosegmentowe wyświetlacze LCD są jasne, zatem odczytywanie prezentowanych wartości nie stanowi żadnego problemu. Na przykład na rysunku 6.14 pokazano, jak wygląda sterowany przez ten szkic moduł LED wyświetlający cyfrę 9 i punkt dziesiętny.

Rysunek 6.14. Cyfra 9 na wyświetlaczu LED

Wyświetlanie dwóch cyfr

Aby użyć większej liczby rejestrów przesuwających do sterowania dodatkowymi wyświetlaczami, należy połączyć pin nr 9 rejestru 74HC595 (otrzymującego dane z płytki Arduino) z pinem nr 14 drugiego rejestru przesuwającego. Po połączeniu obu rejestrów należy wysyłać po 2 bajty danych — pierwszy steruje drugim rejestrem przesuwającym, drugi steruje pierwszym rejestrem przesuwającym. Oto przykład:

```
digitalWrite(LATCH, LOW);
shiftOut(DATA, CLOCK, MSBFIRST, 254); // dane dla drugiego rejestru 74HC595
shiftOut(DATA, CLOCK, MSBFIRST, 254); // dane dla pierwszego rejestru 74HC595
digitalWrite(LATCH, HIGH);
```

Projekt nr 19: sterowanie dwoma modułami wyświetlaczy siedmiosegmentowych LED

Ten projekt ma zademonstrować sposób sterowania dwoma modułami wyświetlaczy siedmiosegmentowych LED wyświetlającymi liczby dwucyfrowe.

Sprzęt

Do realizacji tego projektu będzie potrzebny następujący sprzęt:

- dwa rejestry przesuwające 74HC595;
- dwa siedmiosegmentowe wyświetlacze LED ze wspólnymi katodami;
- szesnaście rezystorów 560 Ω (od R1 do R16);
- duża płytka uniwersalna;
- różne przewody;
- płytka Arduino i przewód USB.

Schemat

Schemat obwodu obejmującego dwa moduły wyświetlaczy LED pokazano na rysunku 6.15.

Rysunek 6.15. Schemat obwodu dla projektu nr 19

Warto zwrócić uwagę na połączenie pinów danych i zegara rejestrów przesuwających pomiędzy sobą i z płytką Arduino. Pin cyfrowy nr 6 na płytce Arduino jest połączony z pierwszym rejestrem przesuwającym, a pin nr 9 tego rejestru jest połączony z pinem nr 14 drugiego rejestru przesuwającego.

Wyświetlanie wartości liczbowych z przedziału od 0 do 99 będzie wymagało bardziej skomplikowanego szkicu. Jeśli wyświetlana liczba jest mniejsza niż 10, wystarczy wysłać tę liczbę i wartość 0, tak aby cyfra na prawym wyświetlaczu reprezentowała odpowiednią wartość i aby lewy moduł wyświetlał 0. Jeśli jednak liczba jest większa niż 10, należy wyodrębnić obie cyfry i wysłać każdą z nich osobno do odpowiedniego rejestru przesuwającego. Proces można znacznie ułatwić, stosując funkcję matematyczną modulo.

Modulo

Modulo to funkcja matematyczna zwracająca resztę z operacji dzielenia. Na przykład operacja 10 modulo 7 jest równa 3 — innymi słowy, reszta z dzielenia 10 przez 7 wynosi 3. W kodzie szkicu operacja modulo jest reprezentowana przez znak procenta (%). Oto prosty przykład użycia operacji modulo w kodzie szkicu:

```
int a = 8;
int b = 3;
c = a % b;
```

W tym przykładzie zmienna c będzie miała wartość 2. Aby wyodrębnić cyfrę, która ma być wyświetlana na prawym module wyświetlacza dwucyfrowego, wystarczy użyć funkcji modulo zwracającej resztę z dzielenia obu cyfr.

Aby zautomatyzować wyświetlanie liczb jedno- i dwucyfrowych, w szkicu dla tego projektu utworzymy funkcję `displayNumber()`. W ramach tej funkcji zostanie użyta operacja modulo do wyodrębnienia cyfr z liczby dwucyfrowej. Aby wyświetlić na przykład liczbę 23, należy najpierw wyodrębnić pierwszą cyfrę, dzieląc 23 przez 10 — w ten sposób otrzymamy cyfrę 2 (część ułamkową wyniku dzielenia można zignorować). Aby wyodrębnić drugą cyfrę, należy zastosować operację 23 modulo 10, której wynikiem będzie 3.

```
// Projekt nr 19 — sterowanie dwoma modułami wyświetlaczy siedmiosegmentowych LED
#define DATA 6        // połączenie z pinem nr 14 rejestru 74HC595
#define LATCH 8       // połączenie z pinem nr 12 rejestru 74HC595
#define CLOCK 10      // połączenie z pinem nr 11 rejestru 74HC595
// definiuje tablicę z dziesiętną reprezentacją segmentów wyświetlacza dla cyfr od 0 do 9
// i liter od A do F (patrz tabela 6.2)
int digits[] = {252, 96, 218, 242, 102, 182, 190, 224, 254, 246, 238, 62,
↪156, 122, 158, 142};
void setup()
{
   pinMode(LATCH, OUTPUT);
   pinMode(CLOCK, OUTPUT);
```

```
   pinMode(DATA, OUTPUT);
}
void displayNumber(int n)
{
   int left, right=0;
❶ if (n < 10)
   {
      digitalWrite(LATCH, LOW);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[n]);
      shiftOut(DATA, CLOCK, LSBFIRST, 0);
      digitalWrite(LATCH, HIGH);
   }
   else if (n >= 10)
   {
❷     right = n % 10; // reszta z dzielenia wyświetlanej liczby przez 10
      left = n / 10; // wynik dzielenia liczby do wyświetlenia przez 10
      digitalWrite(LATCH, LOW);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[right]);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[left]);
      digitalWrite(LATCH, HIGH);
   }
}

❸ void loop()
{
   int i;
   for ( i = 0 ; i < 100 ; i++ )
   {
      displayNumber(i);
      delay(100);
   }
}
```

W wierszu ❶ funkcja sprawdza, czy liczba do wyświetlenia jest mniejsza niż 10. Jeśli tak, wartość (z pustą pierwszą cyfrą) jest od razu wysyłana do rejestrów przesuwających. Jeśli jednak liczba jest większa lub równa 10, funkcja używa operacji modulo i dzielenia ❷ do wyodrębnienia obu cyfr, a następnie wysyła je osobno do obu rejestrów przesuwających. I wreszcie w funkcji `void loop()` ❸ zastosowano pętlę, która wywołuje funkcję `displayNumber()` w celu wyświetlenia liczb od 0 do 99.

Projekt nr 20: budowa termometru cyfrowego

W tym projekcie obwód obejmujący dwucyfrowy wyświetlacz LED (z projektu nr 19) zostanie uzupełniony o czujnik temperatury TMP36 (stosowany wcześniej w rozdziale 4.) — w ten sposób powstanie termometr cyfrowy. Algorytm

dla tego projektu jest bardzo prosty: należy odczytać napięcie przekazywane przez czujnik TMP36 (tak jak w projekcie nr 12) i przekonwertować jego wartość na stopnie Celsjusza.

Sprzęt

Do realizacji tego projektu będzie potrzebny następujący sprzęt:

- obwód dwucyfrowego wyświetlacza LED z projektu nr 19;
- czujnik temperatury TMP36.

Środkową nóżkę czujnika temperatury TMP36 należy podłączyć do pinu analogowego nr 5, lewą nóżkę należy podłączyć do pinu 5 V, a prawą nóżkę do uziemienia (GND).

Szkic

Oto szkic dla tego projektu:

```
// Projekt nr 20 — budowa termometru cyfrowego
#define DATA 6      // połączenie z pinem nr 14 rejestru 74HC595
#define LATCH 8     // połączenie z pinem nr 12 rejestru 74HC595
#define CLOCK 10    // połączenie z pinem nr 11 rejestru 74HC595

int temp = 0;
float voltage = 0;
float celsius = 0;
float sensor = 0;
int digits[]={
   252, 96, 218, 242, 102, 182, 190, 224, 254, 246, 238, 62, 156, 122,
   ↪158, 142};

void setup()
{
   pinMode(LATCH, OUTPUT);
   pinMode(CLOCK, OUTPUT);
   pinMode(DATA, OUTPUT);
}

void displayNumber(int n)
{
   int left, right = 0;
   if (n < 10)
   {
      digitalWrite(LATCH, LOW);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[n]);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[0]);
      digitalWrite(LATCH, HIGH);
   }
```

```
   if (n >= 10)
   {
      right = n % 10;
      left = n / 10;
      digitalWrite(LATCH, LOW);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[right]);
      shiftOut(DATA, CLOCK, LSBFIRST, digits[left]);
      digitalWrite(LATCH, HIGH);
   }
}

void loop()
{
   sensor = analogRead(5);
   voltage = (sensor * 5000) / 1024; // konwertuje wskazanie czujnika na
                                     // miliwolty
   voltage = voltage - 500;          // odejmuje przesunięcie stosowane dla
                                     // napięcia
   celsius = voltage / 10;           // konwertuje miliwolty na stopnie Celsjusza
   temp = int(celsius); // zmienia zmiennoprzecinkową wartość temperatury na
                        // reprezentację całkowitoliczbową
   displayNumber(temp);
   delay(500);
}
```

Szkic dla tego projektu jest dość prosty i zawiera sporo kodu zaczerpniętego z wcześniejszych projektów — funkcja `displayNumber()` pochodzi z projektu nr 19, a kod przetwarzający temperaturę pochodzi z projektu nr 12. Wywołanie funkcji `delay(500)` w przedostatnim wierszu szkicu eliminuje problem zbyt szybkiej zmiany wyświetlanej temperatury.

Moduły wyświetlaczy matrycowych LED

Każdy, komu podobały się eksperymenty z migającymi diodami LED, zapewne pokocha moduły wyświetlaczy matrycowych. *Moduł wyświetlacza matrycowego LED* składa się z wielu wierszy i kolumn diod LED, którymi można sterować zarówno pojedynczo, jak i w ramach pewnych grup. Moduł, który zostanie użyty w kolejnych projektach (patrz rysunek 6.16), oferuje osiem wierszy i osiem kolumn czerwonych diod LED — oznacza to, że mamy do dyspozycji łącznie 64 diody LED.

Rysunek 6.16. Wyświetlacz matrycowy LED

W tym projekcie zostanie zbudowany jeden obwód, który będzie następnie używany z różnymi szkicami generującymi różne efekty.

Schemat obwodu wyświetlacza matrycowego LED

Symbol wyświetlacza matrycowego LED stosowany na schematach obwodów jest dość skomplikowany (patrz rysunek 6.17).

Rysunek 6.17. Symbol wyświetlacza matrycowego LED

Warto zwrócić uwagę na różnice dzielące sposób numerowania wierszy i kolumn diod LED oraz kolejność pinów pod wyświetlaczem matrycowym (patrz numery pinów na rysunku 6.17). Pin nr 1 znajduje się w prawym dolnym rogu na spodzie wyświetlacza. Na rysunku 6.18 widać małą cyfrę 1 wydrukowaną pod tym pinem.

Piny na wyświetlaczu matrycowym LED są numerowane zgodnie z kierunkiem ruchu wskazówek zegara — pin nr 8 znajduje się w lewym dolnym rogu, a pin nr 16 znajduje się w prawym górnym rogu. Do sterowania wyświetlaczem matrycowym LED będą używane dwa rejestry przesuwające 74HC595 w podobny sposób jak w przypadku wyświetlacza dwucyfrowego w projekcie nr 19.

Schemat obwodu dla tego projektu pokazano na rysunku 6.19. Każdy z rezystorów od R1 do R8 ma wartość oporności 560 omów. Jeden rejestr przesuwający steruje wierszami diod LED, drugi rejestr steruje kolumnami diod. Wyświetlacz matrycowy LED (którego nie uwzględniono na schemacie obwodu) jest połączony z pinami wyjściowymi widocznymi w dolnej części schematu — sposób połączenia opisano w tabeli 6.3.

Rysunek 6.18. Piny na spodzie wyświetlacza matrycowego LED z małą cyfrą 1 nadrukowaną pod pinem w prawym dolnym rogu

Rysunek 6.19. Schemat obwodu sterującego wyświetlaczem matrycowym LED

Tabela 6.3. Tabela połączeń wyświetlacza matrycowego z rejestrami przesuwającymi 74HC595

Pin rejestru sterującego wierszami	Pin wyświetlacza	Pin rejestru sterującego kolumnami	Pin wyświetlacza
15	9	15	13
1	14	1	3
2	8	2	4
3	12	3	10
4	1	4	6
5	7	5	11
6	2	6	15
7	5	7	16

Łączenie obwodu

Rejestry przesuwające należy połączyć z wyświetlaczem matrycowym LED zgodnie ze schematem zawartym w tabeli 6.3 (nie wolno zapomnieć o rezystorach pomiędzy pinami wyjściowymi rejestrów przesuwających sterujących wierszami wyświetlacza a pinami tych wierszy).

Czas zaimplementować obsługę wyświetlacza. Rejestr przesuwający oznaczony na rysunku 6.19 słowem Wiersze odpowiada za przekazywanie napięcia do poszczególnych wierszy wyświetlacza matrycowego; rejestr przesuwający oznaczony słowem Kolumny odpowiada za przekazywanie napięcia z poszczególnych kolumn wyświetlacza do pinu GND. Do przetestowania obwodu zostanie użyty prosty szkic w następnym projekcie. Aby włączyć diodę LED, należy wskazać rejestrom przesuwającym piny odpowiedniego wiersza i odpowiedniej kolumny.

Arytmetyka bitowa

Arytmetyka bitowa umożliwia przetwarzanie binarnych reprezentacji zmiennych typów `int` i `byte`. Zaletą operacji bitowych (w porównaniu z działaniami na liczbach w systemie dziesiętnym) jest możliwość łatwiejszego sterowania cyfrowymi pinami wejściowymi i wyjściowymi oraz porównywania binarnych reprezentacji liczb.

Istnieje sześć operatorów bitowych: koniunkcja, alternatywa, alternatywa wykluczająca, negacja, przesunięcie w lewo i przesunięcie w prawo. Wszystkie te operatory zostaną omówione w poniższych punktach. (Mimo że w przykładach dla poszczególnych operatorów będą stosowane liczby binarne, te same operacje można wykorzystać także dla zmiennych typów `int` i `byte`).

Operator koniunkcji bitowej

Operator koniunkcji bitowej (`&`) służy do porównywania bitów składających się na dwie liczby binarne. Jeśli w obu liczbach bity na tej samej pozycji (w tej samej kolumnie) są takie same, odpowiedni bit wynikowy ma wartość 1; jeśli oba bity są różne, odpowiedni bit wynikowy ma wartość 0. Oto przykład zastosowania tego operatora dla trzech zmiennych typu `byte`:

```
byte a = B00110011;
byte b = B01010001;
byte c = 0;
```

Wynikiem następującego porównania:

```
c = a & b;
```

będzie 00010001. Aby lepiej zrozumieć sposób wyznaczania tej wartości, warto zapisać tę operację w formie diagramu:

```
byte a = B00110011;
//        ||||||||
byte b = B01010001;
//        ||||||||
//        00010001 c = a & b; // c ma wartość koniunkcji bitowej zmiennych a i b
```

Operator alternatywy bitowej

Operator alternatywy bitowej (`|`) porównuje dwie liczby binarne, ale zamiast zwracać wartość 1 w przypadku równości bitów na poszczególnych pozycjach, zwraca 1, jeśli *przynajmniej* jedna z wartości w danej kolumnie wynosi 1. Jeśli bity na danej pozycji w obu liczbach mają wartość 0, operator zwraca wartość 0.

W poniższym przykładzie zastosowano te same zmienne co wcześniej:

```
byte a = B00110011;
byte b = B01010001;
byte c = 0;
```

Wynikiem następującego porównania:

```
c = a | b;
```

będzie 01110011.

Operator bitowej alternatywy wykluczającej

Operator bitowej alternatywy wykluczającej (^) zwraca 1, jeśli porównywane bity są różne, oraz 0, jeśli porównywane bity zawierają tę samą wartość.

W poniższym przykładzie ponownie zastosowano te same zmienne:

```
byte a = B00111100;
byte b = B01011010;
byte c = 0;
```

Wynikiem następującego porównania:

```
c = a ^ b;
```

będzie 01100110.

Operator negacji bitowej

Operator negacji bitowej (~) odwraca bity we wszystkich kolumnach — zera są zmieniane na jedynki, a jedynki są zmieniane na zera. Przeanalizujmy teraz prosty przykład, w którym wynik zastosowania operatora negacji dla zmiennej `a` typu `byte` jest przypisywany zmiennej `b` tego samego typu:

```
byte a = B00111100;
byte b = 0;
b = ~a;
```

zmienna `b` zawiera wartość 11000011.

Operatory bitowego przesunięcia w lewo i w prawo

Operatory bitowego przesunięcia w lewo (<<) i bitowego przesunięcia w prawo (>>) przesuwają bity odpowiednio w lewo i prawo o określoną liczbę pozycji. Jeśli na przykład wartość zmiennej a jest przesuwana w lewo o cztery pozycje:

```
byte a = B00100101;
byte b = a << 4;
```

wartość wynikowa (przypisana zmiennej `b`) będzie równa 01010000. Bity zmiennej `a` są przesuwane o cztery pozycje w lewo, a puste pozycje są wypełniane zerami.

Zastosowanie przesunięcia w przeciwnym kierunku:

```
byte a = B11110001;
byte b = a >> 4;
```

spowoduje, że zmienna `b` będzie miała wartość 00001111.

Projekt nr 21: tworzenie wyświetlacza matrycowego LED

Celem tego projektu jest prezentacja sposobu stosowania wyświetlacza matrycowego LED — w tym projekcie zostanie włączony co drugi wiersz i co druga kolumna diod LED (patrz rysunek 6.20).

Rysunek 6.20. Schemat podświetlenia wyświetlacza matrycowego LED

Aby wyświetlić ten wzorzec, należy wysłać wartość B10101010 do rejestru przesuwającego sterującego wierszami i wartość B10101010 do rejestru przesuwającego sterującego kolumnami. Jedynki i zera w obu tych bajtach reprezentują stan odpowiednich wierszy i kolumn wyświetlacza matrycowego.

UWAGA *Warto zwrócić uwagę na zastosowanie operatora negacji bitowej (~) dla bajta kolumn. Rejestr przesuwający sterujący kolumnami włącza diodę LED, jeśli na odpowiedniej pozycji otrzymuje bit 0. Okazuje się jednak, że rejestr przesuwający sterujący wierszami włącza diodę LED, jeśli na odpowiedniej pozycji otrzymuje bit z wartością 1. W tej sytuacji wystarczy zastosować operator negacji bitowej, aby odwrócić bity bajta wysyłanego do drugiego rejestru przesuwającego (sterującego kolumnami).*

Do wygenerowania efektu pokazanego na rysunku 6.20 zostanie użyty następujący szkic:

```
// Projekt nr 21 — tworzenie wyświetlacza matrycowego LED
#define DATA 6      // połączenie z pinem nr 14 rejestru 74HC595
#define LATCH 8     // połączenie z pinem nr 12 rejestru 74HC595
#define CLOCK 10    // połączenie z pinem nr 11 rejestru 74HC595
```

```
void setup()
{
   pinMode(LATCH, OUTPUT);
   pinMode(CLOCK, OUTPUT);
   pinMode(DATA, OUTPUT);
}

void loop()
{
   digitalWrite(LATCH, LOW);
   shiftOut(DATA, CLOCK, MSBFIRST, ~B10101010); // kolumny
   shiftOut(DATA, CLOCK, MSBFIRST,  B10101010); // wiersze
   digitalWrite(LATCH, HIGH);
   do {} while (1); // nie podejmuje żadnych działań
}
```

Wynik działania tego szkicu pokazano na rysunku 6.21. Szkic spowodował włączenie co drugiej diody LED na wyświetlaczu matrycowym, dzięki czemu powstał wzór szachownicy.

Rysunek 6.21. Wynik działania szkicu dla projektu nr 21

Projekt nr 22: wyświetlanie obrazów na wyświetlaczu matrycowym LED

Aby wyświetlić na wyświetlaczu matrycowym LED obraz lub wzór, należy przygotować funkcję włączającą tylko jedną, wskazaną diodę LED. Wyświetlanie obrazu wymaga włączania i wyłączania odpowiednich diod LED na tyle szybko, aby osiągnąć efekt *bezwładności wzroku* (ang. *persistence of vision — POV*). Zjawisko bezwładności wzroku powoduje, że obraz jest widziany przez oko obserwatora przez ułamek sekundy po zniknięciu. Efekt można wykorzystać

do tworzenia niestandardowych obrazów poprzez przeszukiwanie matrycy i bardzo szybkie wyświetlanie tylko po jednym wierszu diod LED. Opisana technika jest bardzo przydatna podczas wyświetlania animacji i danych oraz tworzenia rozmaitych efektów wizualnych.

Techniki sterowania poszczególnymi diodami LED wyświetlacza matrycowego zostaną zademonstrowane w dwóch kolejnych projektach. W poniższym szkicu funkcja `void setLED()` otrzymuje na wejściu numery wiersza i kolumny oraz czas, w którym odpowiednia dioda LED ma być włączona. Szkic cyklicznie włącza po jednej, losowo wybranej diodzie LED.

```
// Projekt nr 22 — wyświetlanie obrazów na wyświetlaczu matrycowym LED
#define DATA 6      // połączenie z pinem nr 14 rejestru 74HC595
#define LATCH 8     // połączenie z pinem nr 12 rejestru 74HC595
#define CLOCK 10    // połączenie z pinem nr 11 rejestru 74HC595

void setup()
{
   pinMode(LATCH, OUTPUT);
   pinMode(CLOCK, OUTPUT);
   pinMode(DATA, OUTPUT);
   randomSeed(analogRead(0));
}

❶ int binary[] = {1, 2, 4, 8, 16, 32, 64, 128};
int r, c = 0;

void setLED(int row, int column, int del)
{
   digitalWrite(LATCH, LOW);
   shiftOut(DATA, CLOCK, MSBFIRST, ~binary[column]); // kolumny
   shiftOut(DATA, CLOCK, MSBFIRST,  binary[row]);    // wiersze
   digitalWrite(LATCH, HIGH);
   delay(del);
}

void loop()
{
   r = random(8);
   c = random(8);
   setLED(r, c, 1); // włącza losowo wybraną diodę na 1 milisekundę
}
```

Zamiast wysyłać liczby binarne bezpośrednio do funkcji `shiftOut()`, aby za jej pomocą włączać i wyłączać poszczególne diody LED, zastosowano tutaj tabelę gotowych wzorców w formie tablicy `int binary[]`❶. Tabela zawiera dziesiętne odpowiedniki poszczególnych bitów składających się na bajty wysyłane do rejestru przesuwającego. Aby włączyć na przykład diodę LED w 4. wierszu i 4. kolumnie, należy wysłać do obu rejestrów przesuwających (z uwzględnieniem

negacji dla bajta kolumn) wartość `binary[3]` (czyli liczbę 8, w systemie binarnym: B00001000). Opisane rozwiązanie jest dość wygodną alternatywą dla konwertowania numerów wierszy i kolumn na format akceptowany przez rejestr przesuwający — zamiast operować na liczbach binarnych, wystarczy wskazać indeks wiersza i kolumny do włączenia.

Zastosowanie zaledwie 1-milisekundowego opóźnienia w tym projekcie spowoduje, że diody LED będą włączane bardzo szybko, co wywoła u odbiorcy wrażenie (wskutek wspomnianego zjawiska bezwładności wzroku), że świeci się więcej niż jedna dioda LED (mimo że jednocześnie jest włączona tylko jedna dioda). Przykład działania tego obwodu pokazano na rysunku 6.22.

Rysunek 6.22. Projekt nr 22 w akcji

Projekt nr 23: wyświetlanie obrazu na wyświetlaczu matrycowym LED

Celem tego projektu jest prezentacja na wyświetlaczu matrycowym LED takiego obrazu jak na rysunku 6.23.

Każdy wiersz można zdefiniować w formie liczby binarnej, a następnie umieścić wszystkie te liczby w tablicy:

```
byte smile[] = {B00000000,
                B00000110,
                B01000110,
                B00000000,
                B00011000,
                B00000000,
                B11000011,
                B01111110};
```

Rysunek 6.23. Przykładowy obraz wyświetlony z wykorzystaniem zjawiska bezwładności wzroku

Łatwo zauważyć, że jedynki w tablicy odpowiadają włączonym diodom LED z rysunku 6.23. Stosowanie liczb binarnych w tej formie znacznie ułatwia generowanie obrazów. Stosując tryby `MSBFIRST` i `LSBFIRST` w wywołaniach funkcji `shiftOut()`, można łatwo odwracać obraz (patrz następny projekt). Do wyświetlania kolejnych wierszy użyto pętli `for` ❶:

```
// Projekt nr 23 — wyświetlanie obrazu na wyświetlaczu matrycowym LED
#define DATA 6      // połączenie z pinem nr 14 rejestru 74HC595
#define LATCH 8     // połączenie z pinem nr 12 rejestru 74HC595
#define CLOCK 10    // połączenie z pinem nr 11 rejestru 74HC595

byte smile[] = {B00000000, B00000110, B01000110, B00000000, B00011000,
B00000000, B11000011, B01111110};
int binary[] = {1, 2, 4, 8, 16, 32, 64, 128};

void setup()
{
   pinMode(LATCH, OUTPUT);
   pinMode(CLOCK, OUTPUT);
   pinMode(DATA, OUTPUT);
}

void loop()
{
   int a;
❶ for ( a = 0 ; a < 8 ; a++ )
   {
      digitalWrite(LATCH, LOW);
      shiftOut(DATA, CLOCK, MSBFIRST, ~smile[a]);  // kolumny
      shiftOut(DATA, CLOCK, LSBFIRST,  binary[a]); // wiersze
```

```
      digitalWrite(LATCH, HIGH);
      delay(1);
   }
}
```

Efekt działania szkicu — wyświetloną buźkę z mrugającym jednym okiem — pokazano na rysunku 6.24.

Rysunek 6.24. Wynik działania projektu nr 23

Projekt nr 24: prezentacja animacji na wyświetlaczu matrycowym LED

Stosując bitowe operacje arytmetyczne, można przewijać w obu kierunkach na wyświetlaczu obraz przygotowany w projekcie nr 23. Obraz uśmiechniętej buzi może być przesuwany na przykład w prawo, zostać ponownie wyświetlony, zmieniony za pomocą przesunięcia o 1 wszystkich bitów wierszy (przy użyciu operatora >>), a następnie ponownie wyświetlony.

Szkic

Oto szkic dla projektu ilustrującego wyświetlanie animacji:

```
// Projekt nr 24 — prezentacja animacji na wyświetlaczu matrycowym LED
#define DATA 6      // połączenie z pinem nr 14 rejestru 74HC595
#define LATCH 8     // połączenie z pinem nr 12 rejestru 74HC595
#define CLOCK 10    // połączenie z pinem nr 11 rejestru 74HC595

byte smile[] = {B00000000, B00000110, B01000110, B00000000, B00011000,
➥B00000000, B11000011, B01111110};
int binary[] = {1, 2, 4, 8, 16, 32, 64, 128};
```

```
void setup()
{
   pinMode(LATCH, OUTPUT);
   pinMode(CLOCK, OUTPUT);
   pinMode(DATA, OUTPUT);
}

void loop()
{
   int a, hold, shift;
❶ for ( shift = 0 ; shift < 9 ; shift++ )
   {
    ❷ for ( hold = 0 ; hold < 25 ; hold++ )
      {
         for ( a = 0 ; a < 8 ; a++ )
         {
            digitalWrite(LATCH, LOW);
         ❸ shiftOut(DATA, CLOCK, MSBFIRST, ~smile[a]>>shift);
           // kolumny
            shiftOut(DATA, CLOCK, LSBFIRST,  binary[a]);       // wiersze
            digitalWrite(LATCH, HIGH);
            delay(1);
         }
      }
   }
}
```

Szkic wyświetla na matrycy obraz przez 25 cykli wyświetlania w pętli `for` ❷. Zmienna `shift` reprezentuje zakres przesunięcia bitowego w prawo. Po każdym wykonaniu pętli zmienna `shift` jest zwiększana o 1 ❶. Cykl wyświetlania jest następnie powtarzany, a obraz jest przesuwany w prawo o jedną kolumnę diod LED. Zmiana wartości `MSBFIRST` na `LSBFIRST` w trzeciej pętli `for` ❸ odwraca kierunek przesuwania obrazu na wyświetlaczu.

Co dalej?

W tym rozdziale wprowadziłem wiele podstawowych technik, które będą bardzo pomocne podczas realizacji przyszłych projektów. Wyświetlacze matrycowe LED są dość wytrzymałe, zatem zachęcam do samodzielnych eksperymentów i sprawdzania najróżniejszych efektów wizualnych. Ponieważ jednak zakres obrazów, które można prezentować na tych wyświetlaczach, jest dość ograniczony, w następnym rozdziale wprowadzę metody wizualizacji dużo bardziej precyzyjnych danych, w tym tekstu i grafiki.

7

Wyświetlacze ciekłokrystaliczne

W tym rozdziale:

- nauczysz się wyświetlać dane tekstowe i numeryczne na znakowych wyświetlaczach LCD;
- utworzysz znaki niestandardowe wyświetlane na znakowych wyświetlaczach LCD;
- użyjesz dużych graficznych modułów LCD do wyświetlania tekstu i danych;
- opracujesz wyświetlacz prezentujący graficzny wykres z historią temperatur.

Niektóre projekty wymagają wyświetlania informacji adresowanych do użytkownika w inny sposób niż na monitorze zwykłego komputera. Jedną z najprostszych i najbardziej uniwersalnych metod wyświetlania informacji jest użycie modułu LCD (od ang. *liquid crystal display*) połączonego z płytką Arduino. Znakowe moduły LCD umożliwiają wyświetlanie tekstu, znaków niestandardowych i danych liczbowych, natomiast graficzne moduły LCD umożliwiają wyświetlanie grafiki.

Znakowe moduły LCD

Moduły LCD wyświetlające znaki, czyli litery i cyfry, należą do najtańszych i najprostszych spośród wszystkich urządzeń produkowanych w technologii wyświetlaczy ciekłokrystalicznych. Na rynku są dostępne wyświetlacze różnych wielkości, które zwykle są wyrażane w liczbie wyświetlanych wierszy i kolumn znaków. Niektóre wyświetlacze są dodatkowo podświetlane, zatem wybór obejmuje zarówno różne kolory znaków, jak i różne kolory tła. Z platformą Arduino powinien współpracować każdy wyświetlacz LCD z interfejsem zgodnym ze standardem HD44780 lub KS0066. W pierwszym projekcie zostanie użyty wyświetlacz LCD o wymiarach 16×2 (wyświetlający dwa wiersze po 16 znaków); patrz rysunek 7.1.

Rysunek 7.1. Przykład wyświetlacza LCD z trimpotem i złączem pinowym

Trimpot (czyli miniaturowy rezystor nastawny dla tego wyświetlacza LCD) ma wartość oporu równą 10 kΩ i służy do ustawiania kontrastu wyświetlania. Złącze pinowe należy przylutować do otworów w górnej części płytki wyświetlacza LCD, aby ułatwić montaż tego wyświetlacza na płytce uniwersalnej. Otwory na płytce wyświetlacza LCD oznaczono liczbami od 1 do 16. Otwór nr 1 (bliżej narożnika modułu), który oznaczono symbolem VSS, należy połączyć z uziemieniem (patrz schemat pokazany na rysunku 7.2). Do przedstawionego schematu będziemy się odwoływać przy okazji omawiania wszystkich przykładów wykorzystujących ten wyświetlacz LCD.

Rysunek 7.2. Schemat łączenia prostego wyświetlacza LCD

Obsługa znakowego modułu LCD w szkicu

Przed przystąpieniem do stosowania znakowego modułu LCD z rysunku 7.1 warto wyjaśnić działanie niezbędnych funkcji na prostym przykładzie. Należy najpierw napisać i wysłać na płytkę Arduino prosty szkic z listingu 7.1.

Listing 7.1. Szkic demonstrujący działanie znakowego modułu LCD

```
// Listing 7.1
#include <LiquidCrystal.h>
LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // piny RS, E, DB4, DB5, DB6, DB7
                                     // modułu LCD
void setup()
{
   lcd.begin(16, 2);
   lcd.clear();
}
void loop()
{
   lcd.setCursor(5, 0);
   lcd.print("Witaj");
   lcd.setCursor(6, 1);
   lcd.print("swiecie");
   delay(10000);
}
```

Wynik wykonania szkicu z listingu 7.1 pokazano na rysunku 7.3.

Rysunek 7.3. Prosty przykład możliwości znakowego modułu LCD

Czas przeanalizować działanie szkicu z listingu 7.1. Szkic musi zawierać na początku dwa wiersze. Ich celem jest odpowiednio dołączenie do szkicu biblioteki obsługującej moduły LCD (która jest automatycznie instalowana wraz ze środowiskiem Arduino IDE) oraz wskazanie tej bibliotece pinów połączonych z płytką Arduino. Oba te zadania zrealizowano, umieszczając *przed* funkcją `void setup()` następujące wiersze:

```
#include <LiquidCrystal.h>
LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // piny RS, E, DB4, DB5, DB6, DB7
                                     // modułu LCD
```

W razie konieczności wykorzystania innych pinów cyfrowych na płytce Arduino wystarczy odpowiednio zmienić numery pinów w drugim z tych wierszy kodu.

W ciele funkcji `void setup()` została określona wielkość modułu LCD wyrażona w liczbie kolumn i wierszy. Na przykład poniższy wiersz sygnalizuje systemowi Arduino, że moduł LCD ma dwa wiersze po 16 znaków:

```
lcd.begin(16, 2);
```

Wyświetlanie tekstu

Po zakończeniu konfiguracji modułu LCD należy wyzerować stan tego wyświetlacza za pomocą następującego wiersza:

```
lcd.clear();
```

Następnie należy określić pozycję kursora, czyli punkt początkowy dla wyświetlanego tekstu:

```
lcd.setCursor(x, y);
```

Wartość x reprezentuje kolumnę (od 0 do 15), wartość y reprezentuje wiersz (0 lub 1). Aby na przykład wyświetlić słowo tekst, wystarczy zastosować w kodzie szkicu następujące wyrażenie:

```
lcd.print("tekst");
```

Skoro wyjaśniłem już, jak określać pozycję tekstu i jak wyświetlać ten tekst, czas omówić wyświetlanie zmiennych danych.

Wyświetlanie zmiennych i liczb

Aby wyświetlić zawartość zmiennej na ekranie modułu LCD, należy użyć następującego wiersza:

```
lcd.print(variable);
```

W przypadku wyświetlania wartości zmiennej zmiennoprzecinkowej istnieje możliwość dodatkowego określenia liczby miejsc po przecinku. Na przykład wyrażenie `lcd.print(pi, 3)` w poniższym przykładzie informuje system Arduino, że należy wyświetlić wartość zmiennej pi z trzema miejscami po przecinku (patrz rysunek 7.4):

```
float pi = 3.141592654;
lcd.print("pi: ");
lcd.print(pi, 3);
```

Rysunek 7.4. Znakowy moduł LCD wyświetlający liczbę zmiennoprzecinkową

Liczbę całkowitą można wyświetlić na ekranie LCD zarówno w systemie dziesiętnym, jak i w formie odpowiedniej reprezentacji szesnastkowej lub binarnej (patrz listing 7.2).

Listing 7.2. Funkcje wyświetlające binarne i szesnastkowe reprezentacje liczb

```
// Listing 7.2
int zz = 170;
lcd.setCursor(0, 0);
lcd.print("Binarna ");
lcd.print(zz, BIN);  // wyświetla reprezentację binarną liczby 170
```

```
lcd.setCursor(0, 1);
lcd.print("Szesnastkowa: ");
lcd.print(zz, HEX);    // wyświetla reprezentację szesnastkową liczby 170
```

Po wykonaniu tego kodu na ekranie LCD zostanie wyświetlony tekst taki jak na rysunku 7.5.

Rysunek 7.5. Wynik wykonania kodu z listingu 7.2

Projekt nr 25: definiowanie znaków niestandardowych

Oprócz standardowych liter, cyfr i symboli dostępnych na klawiaturze istnieje możliwość zdefiniowania maksymalnie ośmiu własnych znaków w ramach jednego szkicu. Łatwo zauważyć, że każdy znak wyświetlany przez moduł LCD składa się z ośmiu wierszy po pięć punktów (*pikseli*). Na rysunku 7.6 wyraźnie widać strukturę poszczególnych znaków.

Rysunek 7.6. Każdy znak składa się z ośmiu wierszy po pięć pikseli

Aby wyświetlać własne znaki, należy najpierw określić, które piksele w poszczególnych wierszach mają być podświetlone. By na przykład utworzyć symbol uśmiechniętej buzi, wystarczy zdefiniować następującą tablicę:

```
byte a[8] = { B00000,
              B01010,
              B01010,
              B00000,
              B00100,
              B10001,
              B01110,
              B00000 };
```

Każda cyfra w tej tablicy opisuje stan określonego piksela na wyświetlaczu LCD. Cyfra 0 wyłącza piksel, natomiast cyfra 1 włącza piksel. Poszczególne elementy tej tablicy reprezentują wiersze pikseli wyświetlacza; pierwszy element reprezentuje stan najwyższego wiersza, drugi element reprezentuje stan drugiego wiersza itd.

UWAGA *Podczas projektowania znaków niestandardowych warto wykorzystać zwykły papier milimetrowy. Każdy wypełniony kwadrat reprezentuje wartość 1, a każdy pusty kwadrat reprezentuje cyfrę 0 w tablicy.*

W tym przypadku pierwszy element ma wartość `B00000`, zatem wszystkie piksele najwyższego wiersza są wyłączone (element zawiera same zera). Następny element (wartość `B01010`) spowoduje włączenie co drugiego piksela — jedynki reprezentują górną część oczu. Każdy kolejny wiersz i piksel dopełnia ten niestandardowy symbol.

Tablicę definiującą ten nowy znak należy następnie przypisać do pierwszego z ośmiu dostępnych miejsc na znaki niestandardowe. W tym celu trzeba wywołać następującą funkcję w ciele funkcji `void setup()`:

```
lcd.createChar(0, a); // przypisuje tablicę a[8] do pierwszego miejsca na znaki
                      // niestandardowe (oznaczonego numerem 0)
```

I wreszcie aby wyświetlić ten znak, należy umieścić następujący wiersz w funkcji `void loop()`:

```
lcd.write(byte(0));
```

Do wyświetlania tego znaku niestandardowego zostanie użyty następujący kod:

```
// Projekt nr 25 — definiowanie znaków niestandardowych
#include <LiquidCrystal.h>
LiquidCrystal lcd(4, 5, 6, 7, 8, 9); // piny RS, E, DB4, DB5, DB6, DB7
                                     // modułu LCD
byte a[8] = { B00000,
              B01010,
              B01010,
              B00000,
              B00100,
              B10001,
              B01110,
              B00000 };
void setup()
{
  lcd.createChar(0, a);
  lcd.begin(16, 2);
```

```
}
void loop()
{
   lcd.write(byte(0)); // wyświetla znak niestandardowy nr 0
                       // w miejscu ustawienia kursora
}
```

Na rysunku 7.7 pokazano uśmiechnięte buzie wyświetlone przez znakowy moduł LCD.

Rysunek 7.7. Wynik działania szkicu dla projektu nr 25

Znakowe moduły LCD są proste w użyciu i oferują dość duże możliwości. Opisane tutaj techniki w zupełności wystarczą na przykład do zbudowania precyzyjnego termometru cyfrowego poprzez połączenie modułu LCD z czujnikiem temperatury opracowanym w ramach projektu nr 20. Jeśli jednak realizowany projekt wymaga wyświetlania dużej ilości danych lub elementów graficznych, należy użyć *graficznego modułu LCD*.

Graficzne moduły LCD

Graficzne moduły LCD są co prawda większe i droższe od modułów znakowych, ale też oferują nieporównanie większe możliwości. Moduły tego typu mogą służyć zarówno do wyświetlania tekstu, jak i do rysowania linii, kropek, okręgów oraz tworzenia rozmaitych efektów wizualnych. W tej książce będzie stosowany graficzny moduł LCD o wymiarach 128 na 64 piksele z interfejsem zgodnym ze standardem KS0108B (patrz rysunek 7.8).

Rysunek 7.8. Graficzny moduł LCD

Tak jak w przypadku znakowego modułu LCD, trimpot modułu graficznego ma wartość 10 kΩ i służy do ustawiania kontrastu wyświetlacza. Złącze pinowe jest przylutowane do wiersza otworów wzdłuż dolnej krawędzi modułu LCD, dzięki czemu montaż na płytce uniwersalnej jest dużo wygodniejszy. Otwory oznaczono numerami od 1 do 20. Pin nr 1 to ten znajdujący się najbliżej narożnika modułu.

Łączenie graficznego modułu LCD

Zanim będzie możliwe użycie graficznego modułu LCD, należy podłączyć 20 przewodów pomiędzy tym modułem a płytką Arduino. Oba komponenty należy połączyć zgodnie z opisem w tabeli 7.1.

Tabela 7.1. Połączenia graficznego modułu LCD z płytką Arduino

Numer pinu modułu LCD	Pin na płytce Arduino	Funkcja pinu modułu LCD
1	5 V	VDD
2	GND	VSS (GND)
3	Środek trimpota 10 kΩ	VO (kontrast)
4	D8	DB0
5	D9	DB1
6	D10	DB2
7	D11	DB3
8	D4	DB4
9	D5	DB5
10	D6	DB6
11	D7	DB7
12	A0	CS2
13	A1	CS1
14	RST	RESET
15	A2	R/W
16	A3	D/I
17	A4	E
18	Zewnętrzna nóżka trimpota, drugą nóżkę należy połączyć z zasilaniem 5 V	VEE (zasilanie trimpota)
19	5 V przez rezystor 22 Ω	Anoda diody LED podświetlenia (+)
20	GND	Katoda diody LED podświetlenia (–)

Stosowanie modułu LCD

Po podłączeniu wyświetlacza należy pobrać i zainstalować bibliotekę dla platformy Arduino niezbędną do obsługi graficznego modułu LCD. Najnowszą wersję tej biblioteki można pobrać ze strony *http://code.google.com/p/glcd-arduino/downloads/list*. Bibliotekę należy zainstalować zgodnie z procedurą opisaną w podrozdziale „Rozszerzanie szkiców za pomocą bibliotek” w rozdziale 8.

Aby korzystanie z modułu LCD było możliwe, w kodzie szkicu przed funkcją `void setup()` należy umieścić następujące wiersze:

```
#include <glcd.h>                // dołącza bibliotekę dla graficznego modułu LCD
#include "fonts/SystemFont5x7.h" // dołącza standardowe czcionki dla tego modułu
```

W ciele funkcji `void setup()` należy dodać następujące wiersze odpowiedzialne za przygotowanie wyświetlacza:

```
GLCD.Init(NON_INVERTED);      // wartość INVERTED spowodowałaby odwrócenie
                              // stanu pikseli (włączonych lub wyłączonych)
GLCD.SelectFont(System5x7);   // wybór czcionki stosowanej na wyświetlaczu
GLCD.ClearScreen();           // zerowanie ekranu LCD
```

Sterowanie wyświetlaczem

Graficzny moduł LCD wyświetla maksymalnie osiem wierszy po 20 znaków tekstu. Aby ustawić pozycję kursora tekstowego, należy użyć następującego wyrażenia, zastępując parametry *x* i *y* właściwymi współrzędnymi:

```
GLCD.CursorTo(x, y);
```

Aby wyświetlić określony tekst, należy zastąpić słowo *tekst* w poniższym przykładzie tekstem do wyświetlenia:

```
GLCD.Puts("tekst");
```

By wyświetlić liczbę całkowitą, należy użyć następującego wiersza (parametr *liczba* trzeba zastąpić liczbą do wyświetlenia):

```
GLCD.PrintNumber(liczba);
```

Projekt nr 26: funkcje tekstowe w praktyce

Ten szkic demonstruje działanie funkcji tekstowych:

```
// Projekt nr 26 — funkcje tekstowe w praktyce
#include <glcd.h>                // dołącza bibliotekę dla graficznego modułu LCD
#include "fonts/SystemFont5x7.h" // dołącza standardowe czcionki dla tego modułu
```

```
int j = 7;

void setup()
{
   GLCD.Init(NON_INVERTED);
   GLCD.ClearScreen();
   GLCD.SelectFont(System5x7);
}

void loop()
{
   GLCD.ClearScreen();
   GLCD.CursorTo(1, 1);
   GLCD.Puts("Witaj swiecie.");
   GLCD.CursorTo(1, 2);
   GLCD.Puts("Mam nadzieje ze");
   GLCD.CursorTo(1, 3);
   GLCD.Puts("podoba Ci sie ta");
   GLCD.CursorTo(1, 4);
   GLCD.Puts("ksiazka. ");
   GLCD.CursorTo(1, 5);
   GLCD.Puts("Ten tekst jest z");
   GLCD.CursorTo(1, 6);
   GLCD.Puts("rozdzialu");
   GLCD.PrintNumber(j);
   GLCD.Puts(".");
   do {} while (1);
}
```

Szkic powinien wyświetlić dane wynikowe pokazane na rysunku 7.9.

Rysunek 7.9. Dane wynikowe szkicu z projektu nr 26

Tworzenie złożonych efektów wizualnych

Przeanalizujmy teraz kilka funkcji umożliwiających tworzenie rozmaitych efektów graficznych. Warto podkreślić, że rozdzielczość stosowanego ekranu LCD wynosi 128 kolumn po 64 piksele, jednak piksele są numerowane od zera, zatem w odwołaniach do nich należy stosować wartości odpowiednio od 0 do 127 i od 0 do 63.

Ta funkcja graficzna włącza pojedynczy piksel o współrzędnych *x*, *y* (jeśli parametr *kolor* ma wartość BLACK) lub wyłącza pojedynczy piksel wyświetlacza (jeśli parametr *kolor* ma wartość WHITE). Kolor czarny (BLACK) zawsze oznacza włączenie piksela, a kolor biały (WHITE) zawsze oznacza jego wyłączenie.

```
GLCD.SetDot(x, y, kolor); // parametr kolor musi mieć wartość BLACK lub WHITE
```

Następna funkcja rysuje prostokąt z lewym górnym narożnikiem w punkcie *x*, *y*. Szerokość prostokąta jest reprezentowana przez parametr *w*, natomiast jego wysokość jest reprezentowana przez parametr *h*. Wyświetlony prostokąt ma postać czarnej ramki z białym wypełnieniem.

```
GLCD.DrawRect(x, y, w, h, kolor);
```

Ta funkcja rysuje wypełniony prostokąt (na podstawie tych samych parametrów):

```
GLCD.FillRect(x, y, w, h, kolor);
```

Ta funkcja rysuje prostokąt z zaokrąglonymi narożnikami (na podstawie tych samych parametrów i parametru *r* reprezentującego promień zaokrąglenia narożników):

```
GLCD.DrawRoundRect(x, y, w, h, r, kolor);
```

Ta funkcja rysuje koło, którego środek znajduje się w punkcie *x*, *y* i którego promień wynosi *r* pikseli:

```
GLCD.DrawCircle(x, y, r, kolor);
```

Ta funkcja rysuje pionowy odcinek, począwszy od punktu *x*, *y*, o długości *l* pikseli:

```
GLCD.DrawVertLine(x, y, l, kolor);
```

A ta funkcja rysuje poziomy odcinek, począwszy od punktu *x*, *y*, o długości *l* pikseli:

```
GLCD.DrawHoriLine(x, y, l, kolor);
```

Odrobina wyobraźni i opisane powyżej funkcje w zupełności wystarczą do tworzenia rozmaitych efektów wizualnych i graficznego prezentowania danych. W następnym podrozdziale zbudujemy szybki termometr wyświetlający wyniki na ekranie LCD przy użyciu części tych funkcji.

Projekt nr 27: budowa szybkiego termometru z wyświetlaną historią

Celem tego projektu jest mierzenie temperatury co 20 minut i wyświetlanie ostatnich 100 pomiarów na wykresie liniowym. Każdy odczyt jest reprezentowany przez jeden piksel; temperatura jest reprezentowana przez oś pionową, a czas przez oś poziomą. Najnowszy odczyt jest widoczny z lewej strony, a wyświetlacz stale przewija wykres ostatnich pomiarów od lewej do prawej strony. Bieżąca temperatura jest dodatkowo wyświetlana w formie liczbowej.

Algorytm

Mimo że na pierwszy rzut oka realizacja tego projektu wydaje się skomplikowana, w rzeczywistości niezbędny szkic wymaga zaledwie dwóch funkcji. Pierwsza z nich odczytuje temperaturę z czujnika TMP36 i zapisuje ją w tablicy zawierającej 100 ostatnich wartości. Za każdym razem, gdy jest odczytywany nowy pomiar, 99 poprzednich odczytów jest przesuwanych w dół tablicy — w ten sposób najstarszy wpis jest usuwany i pojawia się miejsce dla nowego wpisu. Druga funkcja rysuje dane na ekranie LCD. Funkcja odpowiada za wyświetlanie bieżącej temperatury, skali wykresu, wyznaczanie pozycji każdego z pikseli na wykresie i wreszcie wyświetlanie wykresu temperatur.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- graficzny moduł LCD o rozdzielczości 128×64 piksele, z interfejsem KS0108B i pinami przystosowanymi do podłączenia do płytki uniwersalnej;
- trimpot 10 kΩ;
- czujnik temperatury TMP36;
- różne przewody;
- płytka uniwersalna;
- płytka Arduino i przewód USB.

Moduł LCD należy połączyć z płytką Arduino zgodnie z opisem w tabeli 7.1, a czujnik temperatury TMP36 należy połączyć z pinami 5 V, pinem analogowym nr 5 oraz pinem uziemienia (GND).

Szkic

Należy jeszcze wysłać na płytkę Arduino następujący szkic. Kod zawiera komentarze opisujące działanie poszczególnych funkcji.

```
// Projekt nr 27 — budowa szybkiego termometru z wyświetlaną historią

#include <glcd.h>                  // dołącza bibliotekę dla graficznego modułu LCD
#include <fonts/SystemFont5x7.h> // dołącza standardowe czcionki dla tego modułu

int tcurrent;
int tempArray[100];

void setup()
{
   GLCD.Init(NON_INVERTED); // konfiguruje moduł GLCD
   GLCD.ClearScreen();      // wyłącza wszystkie piksele modułu GLCD
   GLCD.SelectFont(System5x7);
}

void getTemp() // funkcja odczytująca temperaturę z czujnika TMP36
{
   float sum = 0;
   float voltage = 0;
   float sensor = 0;
   float celsius;

   // odczytuje temperaturę i konwertuje wynik na stopnie Celsjusza
   sensor   = analogRead(5);
   voltage  = (sensor * 5000) / 1024;
   voltage  = voltage - 500;
   celsius  = voltage / 10;
   tcurrent = int(celsius);

   // umieszcza nową temperaturę na początku tablicy z historią temperatur
   for (int a = 99 ; a >= 0 ; --a )
   {
      tempArray[a] = tempArray[a-1];
   }
   tempArray[0] = tcurrent;
}

void drawScreen() // generuje efekty wizualne na wyświetlaczu GLCD
{
   int q;
   GLCD.ClearScreen();
   GLCD.CursorTo(5, 0);
   GLCD.Puts("Aktualna:");
   GLCD.PrintNumber(tcurrent);
```

```
   GLCD.CursorTo(0, 1);
   GLCD.PrintNumber(40);
   GLCD.CursorTo(0, 2);
   GLCD.PrintNumber(32);
   GLCD.CursorTo(0, 3);
   GLCD.PrintNumber(24);
   GLCD.CursorTo(0, 4);
   GLCD.PrintNumber(16);
   GLCD.CursorTo(1, 5);
   GLCD.PrintNumber(8);
   GLCD.CursorTo(1, 6);
   GLCD.PrintNumber(0);
   for (int a = 28 ; a < 127 ; a++)
   {
      q = (55 - tempArray[a-28]);
      GLCD.SetDot(a, q, BLACK);
   }
}

void loop()
{
   getTemp();
   drawScreen();
   for (int a = 0 ; a < 20 ; a++) // czeka 20 minut przed następnym odczytem
   {
      delay(60000); // czeka 1 minutę
   }
}
```

Wynik

Po uruchomieniu tego szkicu na ekranie LCD powinien zostać wyświetlony obraz podobny do tego z rysunku 7.10.

Rysunek 7.10. Wynik wykonania szkicu dla projektu nr 27

Modyfikowanie szkicu

Wielu osobom interpretacja danych prezentowanych w formie graficznej przychodzi dużo łatwiej niż czytanie liczb. Opisany projekt można by z powodzeniem wykorzystać do wyświetlania także innych rodzajów danych, na przykład napięć przekazywanych przez różne czujniki i mierzonych za pomocą analogowych pinów wejściowych. Istnieje też możliwość dodania jeszcze jednego czujnika temperatury i równoczesnego prezentowania wskazań obu czujników. Niemal wszystko, co zwraca jakąś wartość, można wyświetlić za pomocą graficznego modułu LCD.

Co dalej?

Skoro potrafimy już budować urządzenia z ekranami LCD, możemy wreszcie zacząć postrzegać płytkę Arduino jako niewielki komputer, który może otrzymywać i przetwarzać dane przychodzące, a następnie wyświetlać wyniki tego przetwarzania. To jednak dopiero początek. W następnym rozdziale zbudujemy własne płytki prototypowe dla platformy Arduino, wyjaśnię, jak rejestrować dane na kartach pamięci microSD, i zaprezentuję techniki stosowania bibliotek i funkcji zarządzania czasem wykonywania szkicu.

8

Rozszerzanie możliwości platformy Arduino

W tym rozdziale:

- poznasz rozmaite moduły dodatkowe dla platformy Arduino;
- zbudujesz własny moduł dla platformy Arduino na bazie płytki ProtoShield;
- dowiesz się, jak biblioteki dla platformy Arduino mogą poszerzać zakres dostępnych funkcji;
- użyjesz modułu kart microSD do rejestrowania danych z myślą o analizie w arkuszu kalkulacyjnym;
- zbudujesz urządzenie rejestrujące odczytywaną temperaturę;
- dowiesz się, jak zbudować stoper przy użyciu funkcji `micros()` i `millis()`;
- opanujesz przerwania systemu Arduino i poznasz ich zastosowania.

W tym rozdziale nadal będziemy poszukiwać sposobów poszerzania możliwości oferowanych przez płytkę Arduino (za pomocą różnych modułów dodatkowych) i przeanalizujemy przykład ilustrujący proces budowy własnego modułu. Z czasem, po nabraniu większego doświadczenia w eksperymentach z elektroniką i platformą Arduino, być może zdecydujesz się na budowę trwałych obwodów na płytce *ProtoShield*, czyli pustej płytce drukowanej umożliwiającej konstruowanie własnych, niestandardowych obwodów.

Jednym z najbardziej przydatnych modułów dodatkowych dla platformy Arduino jest moduł kart *microSD*. W tym rozdziale moduł zostanie użyty do budowy urządzenia rejestrującego zmiany temperatury w czasie — moduł umożliwia rejestrowanie danych przez system Arduino i ich późniejsze przenoszenie na inną platformę w celu dalszych analiz.

W tym rozdziale omówię też funkcje `micros()` i `millis()`, które bardzo ułatwiają śledzenie czasu — aby lepiej zilustrować ich działanie, zbuduję projekt stopera. I wreszcie wprowadzę przerwania systemu Arduino.

Moduły

Możliwości płytki Arduino można rozszerzyć, montując na niej tzw. *moduły* (ang. *shields*). Moduł to płytka z obwodem łączona z płytką Arduino za pośrednictwem pinów umieszczanych w odpowiednich gniazdach przy krawędziach oryginalnej platformy. Na rynku są dostępne setki różnych modułów. Jeden z najbardziej popularnych projektów łączy w sobie moduł GPS z modułem kart pamięci microSD — takie połączenie pozwala łatwo stworzyć urządzenie rejestrujące i zapisujące położenie w dłuższym czasie, na przykład drogę przebytą przez samochód lub nową trasę wspinaczkową. Inne projekty obejmują między innymi moduły sieciowe umożliwiające łączenie platformy Arduino z internetem (patrz rysunek 8.1).

Rysunek 8.1. Moduł sieciowy zainstalowany na płytce Arduino Uno

Odbiorniki sygnału satelitarnego GPS pozwalają na śledzenie położenia systemu Arduino (patrz rysunek 8.2). Interfejsy kart pamięci microSD umożliwiają platformie Arduino zapisywanie danych na kartach (patrz rysunek 8.3).

Rysunek 8.2. Moduł odbiornika GPS (z dołączonym modułem GPS) na płytce Arduino Uno

Rysunek 8.3. Zestaw modułu kart pamięci microSD

Moduły dla platformy Arduino są projektowane w taki sposób, aby można było umieszczać jeden moduł nad drugim, ponieważ zwykle stosuje się wiele modułów jednocześnie. Na rysunku 8.4 pokazano przykład takiego połączenia obejmującego płytkę Arduino Uno, moduł kart pamięci microSD (umożliwiający rejestrowanie danych), moduł sieciowy (umożliwiający łączenie systemu z internetem) oraz moduł LCD (wyświetlający informacje).

Rysunek 8.4. Cztery moduły umieszczone jednocześnie na płytce Arduino Uno

OSTRZEŻENIE *Podczas montażu wielu modułów jeden na drugim należy sprawdzić, czy zastosowane moduły nie korzystają z tych samych pinów analogowych lub cyfrowych. Używanie tych samych pinów przez moduły do różnych celów może uszkodzić cały układ, zatem warto zachować daleko idącą ostrożność. Producenci modułów dołączają do swoich urządzeń informacje o używanych pinach platformy Arduino. Doskonałą, stale aktualizowaną przez społeczność listę modułów wraz z używanymi pinami można znaleźć na stronie http://www.shieldlist.org/.*

Płytka prototypowa ProtoShield

Moduły dodatkowe dla platformy Arduino można nie tylko kupić w formie gotowych urządzeń (na przykład w sklepie *http://www.shieldlist.org/*), ale też samodzielnie konstruować za pomocą płytki prototypowej ProtoShield. Płytka ProtoShield jest dostępna w formie zestawu do złożenia (patrz rysunek 8.5) lub w wersji w pełni połączonej.

Rysunek 8.5. Zestaw płytki ProtoShield

Płytka ProtoShield może też stanowić wygodną podstawę dla płytki uniwersalnej, ponieważ pozwala zmieścić niewielki obwód w fizycznych granicach wyznaczonych przez wymiary płytki Arduino (przykład tak skonstruowanego termometru pokazano na rysunku 8.6). Mniejsze płytki uniwersalne można instalować na płytce drukowanej przy użyciu masy klejącej wielokrotnego użytku Blu-Tack (ta forma montażu sprawdza się w przypadku tymczasowych rozwiązań) lub za pomocą dwustronnej taśmy klejącej (w przypadku układów, które mają gwarantować większą trwałość). Płytki ProtoShield mogą też służyć za trwałą podstawę dla obwodów, które wcześniej przetestowano na płytkach uniwersalnych.

Rysunek 8.6. Termometr zbudowany w ramach projektu nr 8

Budowa niestandardowych obwodów na płytce ProtoShield wymaga odpowiedniego planowania. Przed przystąpieniem do właściwego montażu należy zaprojektować obwód, przygotować schemat połączeń oraz zaplanować rozmieszczenie komponentów, tak aby zmieściły się na płytce ProtoShield. I wreszcie gotowy obwód niestandardowego modułu należy przylutować do płytki (ale dopiero po uprzednim przetestowaniu układu na płytce uniwersalnej, ponieważ tylko wówczas możemy mieć pewność co do jego prawidłowego działania). Niektóre płytki ProtoShield są oferowane wraz ze schematem w pliku PDF, który można pobrać i wydrukować, aby następnie łatwiej narysować schemat obwodu.

Projekt nr 28: tworzenie modułu niestandardowego z ośmioma diodami LED

W tym projekcie zbudujemy moduł niestandardowy zawierający osiem diod LED wraz z niezbędnymi rezystorami ograniczającymi napięcie. Moduł niestandardowy w tej formie ułatwi przyszłe eksperymenty z diodami LED i cyfrowymi pinami wyjściowymi.

Sprzęt

Do budowy tego modułu będą potrzebne następujące elementy:

- pusta płytka Arduino ProtoShield;
- osiem diod LED dowolnego koloru;
- osiem rezystorów 560 Ω (od R1 do R8);
- dwa 6-pinowe złącza pasujące do gniazd na płytce Arduino;
- dwa 8-pinowe złącza pasujące do gniazd na płytce Arduino.

Schemat

Schemat obwodu tego modułu pokazano na rysunku 8.7.

Rysunek 8.7. Schemat obwodu dla projektu nr 28

Układ płytki prototypowej ProtoShield

Następnym krokiem jest opanowanie układu otworów na płytce ProtoShield. Wiersze i kolumny otworów na tej płytce powinny pasować do wierszy i kolumn na płytce uniwersalnej. Na pustej płytce ProtoShield (patrz rysunek 8.8) połączone rzędy otworów są oznaczone nadrukowanymi czarnymi liniami.

Długa pozioma linia pod pinami cyfrowymi jest połączona z gniazdem 5 V, natomiast pozioma linia nad pozostałymi pinami i pinami analogowymi jest połączona z uziemieniem (GND). Otwory pomiędzy tymi liniami są od siebie odizolowane. I wreszcie na płytce znajdują się dwa wiersze pinów — jeden na samej górze i drugi na samym dole płytki ProtoShield. Wiersz pinów na górze płytki składa się z dwóch grup po osiem pinów, a wiersz na dole płytki składa

Rysunek 8.8. Górna strona pustej płytki ProtoShield

się z dwóch grup po sześć pinów. Właśnie do tych pinów należy przylutować złącza pinowe, dzięki którym płytkę ProtoShield można łatwo zamontować w gnieździe na płytce Arduino.

Projekt

Projekt obwodu warto narysować na papierze milimetrowym (patrz rysunek 8.9).

Rysunek 8.9. Planowanie budowy modułu niestandardowego

Po narysowaniu planu obwodu na kartce papieru należy testowo (bez lutowania) umieścić komponenty na płytce ProtoShield i sprawdzić, czy mieszczą się w wyznaczonych miejscach i czy nie znajdują się zbyt blisko siebie. Jeśli na płytce ProtoShield jest miejsce na przycisk *RESET*, zawsze należy go zainstalować, ponieważ dodatkowy moduł blokuje dostęp do odpowiedniego przycisku na samej płytce Arduino.

Lutowanie komponentów

Po zaplanowaniu układu komponentów, wstępnym umieszczeniu elementów na płytce ProtoShield i przetestowaniu działania obwodu można przystąpić do lutowania komponentów. Posługiwanie się lutownicą nie jest trudne, a tak proste układy z pewnością nie wymagają zakupu drogiej stacji lutowniczej. Prosta lutownica o mocy 25 – 40 W (podobna do tej pokazanej na rysunku 8.10) powinna w zupełności wystarczyć.

Rysunek 8.10. Lutownica

UWAGA *Czytelników, którzy nie mają doświadczenia z lutowaniem, zachęcam do pobrania i lektury instrukcji w formie komiksu dostępnej na stronie http://mightyohm.com/soldercomic/.*

Podczas lutowania komponentów zdarza się, że należy je połączyć za pomocą niewielkiej ilości cyny lutowniczej (patrz rysunek 8.11). Jak widać, końcówka rezystora jest połączona z anodą diody LED.

Rysunek 8.11. Mostek lutowniczy

Każde lutowane połączenie warto sprawdzić zaraz po zamontowaniu odpowiedniego komponentu, ponieważ ewentualne niedociągnięcia łatwiej zlokalizować i naprawić *przed* zakończeniem całego projektu. Podczas lutowania czterech

złączy pinowych można wykorzystać jakiś istniejący moduł, aby zagwarantować odpowiednie położenie pinów (patrz rysunek 8.12).

Rysunek 8.12. Lutowanie złączy pinowych

Na rysunku 8.13 pokazano gotowy produkt — moduł niestandardowy dla platformy Arduino z ośmioma diodami LED.

Rysunek 8.13. Gotowy moduł niestandardowy!

Modyfikacja modułu niestandardowego

Ten prosty moduł można wykorzystać w ramach bardziej złożonego projektu polegającego na przykład na monitorowaniu stanu pinów cyfrowych od numeru 0 do 7. Dodanie kolejnych sześciu rezystorów i diod LED umożliwiłoby monitorowanie stanu wszystkich pinów cyfrowych. Opisany moduł ma jeszcze mnóstwo innych zastosowań. Wystarczy użyć wyobraźni!

Rozszerzanie szkiców za pomocą bibliotek

Tak jak moduł dodatkowy dla płytki Arduino rozszerza możliwości sprzętowe tej platformy, tak biblioteka dodaje przydatne funkcje realizujące określone zadania lub implementuje obsługę rozmaitych urządzeń — funkcje zdefiniowane w dołączonych bibliotekach można następnie stosować w kodzie szkicu. Bibliotekę może napisać każdy z nas. Producenci modułów dla platformy Arduino często piszą własne biblioteki specjalnie z myślą o obsłudze oferowanego sprzętu.

Środowisko Arduino IDE zawiera już pewien zbiór domyślnie instalowanych bibliotek. Aby dołączyć te biblioteki do pisanych szkiców, wystarczy wybrać opcję *Szkic/Importuj bibliotekę* — zostanie wyświetlona lista zainstalowanych bibliotek, w tym *Ethernet*, *LiquidCrystal*, *Servo* itd. Większość tych nazw nie wymaga dodatkowych wyjaśnień. (Jeśli jakaś biblioteka jest potrzebna do realizacji projektu, wszystkie niezbędne szczegóły wyjaśnię przy okazji omawiania tego projektu).

Importowanie bibliotek dla modułów

Po zakupie modułu zwykle należy pobrać i zainstalować jego biblioteki (dostępne na stronie producenta modułu lub pod innym, wskazanym przez tego producenta adresem).

Aby zademonstrować niezbędną procedurę, pokażę, jak pobrać bibliotekę potrzebną do obsługi modułu kart microSD z rysunku 8.3.

1. Najnowszą wersję pliku należy pobrać ze strony *http://code.google.com/p/sdfatlib/downloads/list*. Wspomnianą stronę internetową pokazano na rysunku 8.14.

Rysunek 8.14. Strona z różnymi wersjami biblioteki

2. Należy kliknąć najnowszą wersję biblioteki na liście plików po lewej stronie, aby przejść na stronę pobierania (patrz rysunek 8.15).

Rysunek 8.15. Strona pobierania biblioteki

3. Aby rozpocząć proces pobierania pliku, należy kliknąć jego nazwę (z rozszerzeniem *.zip*). Po pobraniu biblioteki wystarczy postępować według poniższych wskazówek dotyczących instalacji w różnych systemach operacyjnych.

Instalacja biblioteki w systemie Mac OS X

Jeśli biblioteka została pobrana w systemie Mac OS X, trzeba wykonać następujące kroki:

1. Należy otworzyć folder *Pobrane rzeczy* (*Downloads*) i znaleźć pobrany plik *.zip* (patrz rysunek 8.16).

Rysunek 8.16. Plik biblioteki w folderze Pobrane rzeczy (Downloads)

2. Należy dwukrotnie kliknąć pobrany plik, aby wyświetlić jego zawartość, a następnie znaleźć folder *SdFat* (patrz rysunek 8.17).
3. Folder *SdFat* należy skopiować z okna widocznego na rysunku 8.17 do folderu *Arduino.app/Contents/Resources/libraries* (patrz rysunek 8.18).
4. Warto jeszcze sprawdzić, czy biblioteka *SdFat* została prawidłowo zainstalowana i czy jest dostępna — wystarczy ponownie uruchomić środowisko IDE i wybrać opcję *Szkic/Importuj bibliotekę*. Wyświetlona lista powinna teraz zawierać bibliotekę *SdFat*. (Jeśli biblioteki nie ma na liście, należy ponownie wykonać procedurę instalacji).

Rysunek 8.17. Folder pobranej biblioteki

Rysunek 8.18. Zakończona instalacja biblioteki

Po zainstalowaniu biblioteki można od razu przejść do podrozdziału „Karty pamięci microSD”.

Instalacja biblioteki w systemie Windows XP i nowszych

Jeśli biblioteka została pobrana w systemie Windows XP lub nowszym, należy wykonać następujące kroki:

1. Po zakończeniu pobierania trzeba otworzyć pobrany plik, aby wyświetlić jego zawartość, a następnie znaleźć folder *SdFat* (patrz rysunek 8.19).

Rysunek 8.19. Folder pobranej biblioteki

2. Folder *SdFat* należy skopiować z okna widocznego na rysunku 8.19 do folderu *Arduino/libraries* (patrz rysunek 8.20).

Rysunek 8.20. Zakończona instalacja biblioteki

3. Warto sprawdzić, czy biblioteka *SdFat* została prawidłowo zainstalowana i czy jest dostępna — wystarczy ponownie uruchomić środowisko IDE i wybrać opcję *Szkic/Importuj bibliotekę*. Tym razem lista powinna zawierać bibliotekę *SdFat*. (Jeśli biblioteki nie ma na liście, należy jeszcze raz przeprowadzić procedurę instalacji).

Po zainstalowaniu biblioteki można od razu przejść do podrozdziału „Karty pamięci microSD".

Instalacja biblioteki w systemie Ubuntu Linux 11.04 i nowszych

Jeśli biblioteka została pobrana w systemie Ubuntu Linux 11.04 lub nowszym, trzeba wykonać następujące kroki:

1. Należy znaleźć i dwukrotnie kliknąć pobrany plik. W oknie menedżera archiwum powinien być widoczny folder *SdFat* (patrz rysunek 8.21).
2. Należy dwukrotnie kliknąć folder *SdFat* w oknie pokazanym na rysunku 8.21, a następnie wybrać opcję *Rozpakuj*, aby rozpakować ten folder i umieścić w folderze */libraries* (patrz rysunek 8.22).
3. Warto sprawdzić, czy biblioteka *SdFat* została prawidłowo zainstalowana i czy jest dostępna — wystarczy ponownie uruchomić środowisko IDE i wybrać opcję *Szkic/Importuj bibliotekę*. Wyświetlona lista powinna teraz zawierać bibliotekę *SdFat*. (Jeśli biblioteki nie ma na liście, należy jeszcze raz przeprowadzić procedurę instalacji).

Rysunek 8.21. Folder pobranej biblioteki

Rysunek 8.22. Miejsce docelowe dla folderu biblioteki

Po zainstalowaniu biblioteki można przejść do następnego podrozdziału.

Karty pamięci microSD

Karty microSD umożliwiają rejestrowanie danych zbieranych przez platformę Arduino z wielu różnych źródeł, na przykład za pośrednictwem czujnika temperatury TMP36, który zastosowano w rozdziale 4. Na karcie microSD można też przechowywać dane potrzebne do działania serwera WWW lub dowolne inne pliki wykorzystywane w realizowanym projekcie. Do rejestrowania i przechowywania danych można użyć na przykład modułu kart pamięci microSD podobnego do tego z rysunku 8.3. Pokazany moduł obsługuje karty microSD (nie microSDHC) o maksymalnej pojemności 2 GB.

Rysunek 8.23. Karta microSD o pojemności 2 GB

W popularnych sklepach z elektroniką, jak SparkFun, Adafruit Industries czy Snootlab, można znaleźć wiele różnych modułów obsługujących karty pamięci. Moduł stosowany w tej książce (pokazany na rysunku 8.3) pochodzi ze sklepu SparkFun (produkty oznaczone symbolami DEV-09802 i PRT-10007).

UWAGA *Zanim będzie możliwe stosowanie karty pamięci za pośrednictwem tego modułu, kartę należy sformatować. Aby to zrobić, wystarczy umieścić kartę w czytniku podłączonym do komputera i postępować zgodnie z procedurą formatowania kart pamięci obowiązującą w danym systemie operacyjnym. Podczas formatowania koniecznie należy wybrać system plików FAT16. Warto też pamiętać, że niektóre moduły kart pamięci wymagają lutowania gniazd (patrz punkt „Lutowanie komponentów" we wcześniejszej części tego rozdziału).*

Testowanie karty microSD

Po sformatowaniu i umieszczeniu karty pamięci w gnieździe modułu kart microSD należy jeszcze sprawdzić, czy karta działa prawidłowo. W tym celu wykonaj następujące kroki:

1. Podłącz moduł kart microSD do płytki Arduino, włóż kartę pamięci do gniazda i podłącz przewód USB.
2. Uruchom środowisko IDE i wybierz opcję *Plik/Przykłady/SD/CardInfo*. Po otwarciu szkicu CardInfo w środowisku IDE wyślij program na płytkę Arduino.

3. Otwórz okno monitora portu szeregowego, ustaw przepustowość 9600 oraz naciśnij kolejno dowolny klawisz na klawiaturze i klawisz *Enter*. Po chwili w oknie monitora powinny zostać wyświetlone podstawowe informacje na temat karty microSD (patrz rysunek 8.24).

Rysunek 8.24. Wynik prawidłowo zakończonego testu karty microSD

Jeśli w oknie monitora portu szeregowego nie pojawią się wyniki testu, należy powtarzać poniższą procedurę aż do rozwiązania problemu:

- Odłącz przewód USB od płytki Arduino, usuń kartę microSD z gniazda i ponownie umieść ją w gnieździe kart pamięci.
- Sprawdź, czy złącze pinowe zostało prawidłowo przylutowane do płytki i czy żaden pin nie styka się z sąsiednim pinem.
- Sprawdź, czy w oknie monitora portu szeregowego jest ustawiona przepustowość 9600 i czy stosowana płytka jest zgodna ze standardem Arduino Uno. Model Mega (i kilka innych wersji) ma nieco inaczej rozmieszczone piny SPI.
- Ponownie sformatuj kartę microSD.

Projekt nr 29: zapisywanie danych na karcie pamięci

Aby zapisywać dane na karcie pamięci, należy zamontować odpowiedni moduł na płytce Arduino, umieścić w gnieździe tego modułu kartę microSD, a następnie wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 29 — zapisywanie danych na karcie pamięci

int b = 0;

#include <SD.h>

void setup()
{
   Serial.begin(9600);
   Serial.print("Inicjalizacja karty SD...");
   pinMode(10, OUTPUT);

   // sprawdza, czy karta microSD jest dostępna i gotowa do użycia
   if (!SD.begin(8))
   {
      Serial.println("Brak lub nieprawidłowe działanie karty");
      // zatrzymuje wykonywanie szkicu
      return;
   }
   Serial.println("Karta microSD jest gotowa");
}

void loop()
{
❶ // tworzy plik, w którym będą zapisywane dane
   File dataFile = SD.open("DATA.TXT", FILE_WRITE);
   // jeśli plik jest gotowy, zapisuje dane
   if (dataFile)
❷   {
      for ( int a = 0 ; a < 11 ; a++ )
      {
         dataFile.print(a);
         dataFile.print(" pomnożone przez 2 równa się ");
         b = a * 2;
❸        dataFile.println(b, DEC);
❹      }
❺      dataFile.close(); // po zapisaniu danych zamyka plik
                         // (ten krok jest konieczny)
   }
   // jeśli plik nie jest gotowy, wyświetla komunikat o błędzie
   else
   {
      Serial.println("Nie udało się otworzyć pliku DATA.TXT");
❻ }
   Serial.println("koniec");
   do {} while (1);
}
```

Powyższy szkic tworzy na karcie microSD plik tekstowy nazwany *DATA.txt* (patrz rysunek 8.25).

Rysunek 8.25. Dane wynikowe szkicu z projektu nr 29

Warto bliżej przeanalizować funkcję `void loop()` tego szkicu, aby zrozumieć sposób tworzenia pliku tekstowego. Kod w ciele funkcji `void loop()` pomiędzy wierszami ❶ i ❷ oraz ❹ i ❻ tworzy i otwiera plik w trybie zapisu. Do zapisywania tekstu w tym pliku użyto następujących wywołań:

`dataFile.print();` lub `dataFile.println();`

Wspomniane funkcje działają bardzo podobnie na przykład do funkcji `Serial.println()`, zatem zapisywanie danych w pliku wymaga takich samych działań jak wyświetlanie danych w oknie monitora portu szeregowego. W wierszu ❶ ustawiono nazwę tworzonego pliku tekstowego — nazwa pliku składa się z maksymalnie ośmiu znaków, po których następuje kropka i kolejne trzy znaki tzw. rozszerzenia, na przykład *DATA.txt*.

W wierszu ❸ przekazano wartość `DEC` za pośrednictwem drugiego parametru. Wartość `DEC` oznacza, że zmienna zawiera liczbę dziesiętną i że należy zapisać odpowiednią wartość w pliku tekstowym właśnie w tej formie. Gdyby szkic zapisywał wartość w zmiennej typu `float`, należałoby przekazać cyfrę reprezentującą liczbę miejsc po przecinku (maksymalnie sześciu).

Po zakończeniu zapisywania danych w pliku tekstowym zastosowaliśmy funkcję `dataFile.close()`❺, aby zamknąć ten plik. Gdyby szkic nie zawierał tego wywołania, komputer nie mógłby odczytać pliku tekstowego utworzonego przez nasz szkic.

Projekt nr 30: budowa urządzenia rejestrującego temperaturę

Skoro wyjaśniłem już, jak rejestrować dane, spróbujmy zbudować urządzenie, które będzie przez osiem godzin co minutę mierzyło temperaturę (za pomocą czujnika temperatury TMP36 wprowadzonego w rozdziale 4.) i rejestrowało te odczyty na karcie microSD. W tym projekcie zostaną połączone funkcje zapisujące dane na karcie pamięci microSD (z projektu nr 29) z funkcjami odpowiedzialnymi za mierzenie temperatury (z projektu nr 27).

Sprzęt

Realizacja tego projektu będzie wymagała następującego sprzętu:

- czujnika temperatury TMP36;
- płytki uniwersalnej;
- różnych przewodów;
- modułu kart pamięci microSD i karty;
- płytki Arduino i przewodu USB.

Kartę microSD należy umieścić w gnieździe modułu kart pamięci, a sam moduł należy zainstalować na płytce Arduino. Lewy pin (5 V) czujnika TMP36 trzeba połączyć z pinem 5 V na płytce Arduino, środkowy pin z pinem analogowym, a prawy pin z pinem uziemienia (GND).

Szkic

W środowisku Arduino IDE należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 30 — budowa urządzenia rejestrującego temperaturę

#include <SD.h>
float sensor, voltage, celsius;

void setup()
{
   Serial.begin(9600);
   Serial.print("Inicjalizacja karty SD...");
   pinMode(10, OUTPUT);

   // sprawdza, czy karta microSD jest dostępna i gotowa do użycia
   if (!SD.begin(8))
   {
      Serial.println("Brak lub nieprawidłowe działanie karty");
      // zatrzymuje wykonywanie szkicu
```

```
      return;
   }
   Serial.println("Karta microSD jest gotowa");
}

void loop()
{
   // tworzy plik, w którym będą zapisywane dane
   File dataFile = SD.open("DATA.TXT", FILE_WRITE);
   // jeśli plik jest gotowy, można zapisać dane
   if (dataFile)
   {
      for ( int a = 0 ; a < 481 ; a++ ) // 8 godzin to 480 minut
      {
         sensor = analogRead(0);
         voltage = (sensor * 5000) / 1024; // konwertuje wskazanie czujnika na
                                           // miliwolty
         voltage = voltage - 500;
         celsius = voltage / 10;
         dataFile.print(" Zapis temperatury: ");
         dataFile.print(a, DEC);
         dataFile.print(" Temperatura: ");
         dataFile.print(celsius, 2);
         dataFile.println(" stopni Celsjusza");
         delay(599900); // czeka trochę poniżej jednej minuty
      }
      dataFile.close(); // zamknięcie pliku jest konieczne
      Serial.println("Koniec!");
      do {} while (1);
   }
}
```

Wykonanie pełnego zadania przez ten szkic zajmie nieco ponad 8 godzin, jednak można ten czas łatwo skrócić, zmniejszając wartość przekazywaną w wywołaniu `delay(599900)`.

Po zakończeniu działania szkicu należy wyjąć kartę microSD z gniazda modułu kart pamięci i otworzyć zapisany plik w edytorze tekstu (patrz rysunek 8.26).

Aby przeprowadzić zaawansowaną analizę zgromadzonych danych, wystarczy oddzielić poszczególne kolumny spacjami lub dwukropkami, dzięki czemu dane będzie można łatwo zaimportować do arkusza kalkulacyjnego. Plik można zaimportować na przykład do arkusza OpenOffice Calc, wybierając opcję *Wstaw/Arkusz z pliku* — po wskazaniu pliku zostanie utworzony arkusz podobny do tego pokazanego na rysunku 8.27. Tak zaimportowane dane można poddać dalszej analizie (patrz rysunek 8.28).

Rysunek 8.26. Dane wynikowe wygenerowane przez projekt nr 30

Rysunek 8.27. Importowanie danych do arkusza kalkulacyjnego

Rysunek 8.28. Analiza temperatur

Przykłady demonstrujące rejestrowanie temperatur można łatwo dostosować na potrzeby projektów polegających na gromadzeniu dowolnych innych danych. Te same techniki umożliwiają rejestrowanie każdego rodzaju danych generowanych przez system Arduino.

Zarządzanie czasem wykonywania aplikacji za pomocą funkcji millis() i micros()

Za każdym razem, gdy system Arduino rozpoczyna wykonywanie jakiegoś szkicu, dodatkowo jest rejestrowany czas wykonywania tego szkicu (zarówno w milisekundach, jak i w mikrosekundach). Milisekunda to jedna tysięczna sekundy (0,001), natomiast mikrosekunda to jedna milionowa sekundy (0,000001). Obie wartości umożliwiają mierzenie czasu, który upłynął od momentu uruchomienia szkicu.

Dostęp do tych wartości można uzyskać za pomocą następujących funkcji (zwracających wartości typu `unsigned long`):

```
unsigned long a,b;
a = micros();
b = millis();
```

Z uwagi na ograniczenia typu danych `unsigned long` wartość zwracana przez obie funkcje jest zerowana po osiągnięciu liczby 4 294 967 295, zatem funkcja `millis()` umożliwia śledzenie czasu przez około 50 dni, natomiast funkcja `micros()` pozwala na śledzenie czasu przez około 70 minut. Co więcej, ograniczenia samego mikroprocesora na płytce Arduino powodują, że wartości zwracane przez funkcję `micros()` zawsze są wielokrotnościami liczby 4.

Spróbujmy teraz wykorzystać te wartości do sprawdzenia, ile czasu platforma Arduino potrzebuje na przełączenie pinu cyfrowego ze stanu `LOW` na `HIGH` i odwrotnie. W tym celu wystarczy odczytać wartość funkcji `micros()` przed wywołaniem funkcji `digitalWrite()`i po nim oraz wyświetlić różnicę obu wartości w oknie monitora portu szeregowego. W tym przypadku jedynym wymaganym sprzętem jest płytka Arduino i przewód USB.

Szkic z listingu 8.1 należy wpisać w środowisku Arduino IDE i wysłać na płytkę.

Listing 8.1. Sprawdzanie czasu zmiany stanu pinu cyfrowego za pomocą funkcji micros()

```
// Listing 8.1

unsigned long start, finished, elapsed;

void setup()
{
   Serial.begin(9600);
   pinMode(3, OUTPUT);
   digitalWrite(3, LOW);
}

void loop()
{
```

```
❶ start = micros();
   digitalWrite(3, HIGH);
❷ finished = micros();
❸ elapsed = finished – start;
   Serial.print("LOW -> HIGH: ");
   Serial.print(elapsed);
   Serial.println(" mikrosekund");
   delay(1000);

❹ start = micros();
   digitalWrite(3, LOW);
   finished = micros();
   elapsed = finished - start;
   Serial.print("HIGH -> LOW: ");
   Serial.print(elapsed);
   Serial.println(" mikrosekund");
   delay(1000);
}
```

Szkic sprawdza wartości zwracane przez funkcję `micros()` bezpośrednio przed wywołaniem funkcji `digitalWrite(HIGH)` i po nim (wiersze ❶ i ❷), a następnie oblicza różnicę i wyświetla wynik w oknie monitora portu szeregowego ❸. Procedura jest powtarzana dla odwrotnej zmiany stanu pinu cyfrowego ❹.

Należy teraz otworzyć okno monitora portu szeregowego i sprawdzić wyniki (patrz rysunek 8.29).

Rysunek 8.29. Dane wynikowe szkicu z listingu 8.1

Ponieważ funkcja `micros()` zwraca wielokrotność liczby 4, wartość 8 mikrosekund oznacza, że czas przełączania stanu pinu cyfrowego był większy niż 4 mikrosekundy i nie dłuższy niż 8 mikrosekund.

Projekt nr 31: budowa stopera

Skoro pokazałem już, jak mierzyć czas pomiędzy dwoma zdarzeniami, możemy przystąpić do tworzenia prostego stopera na bazie platformy Arduino. Stoper będzie udostępniał dwa przyciski — jeden będzie służył do uruchamiania i zerowania pomiaru, drugi umożliwi zatrzymanie pomiaru i wyświetlenie zmierzonego czasu. Szkic będzie stale sprawdzał stan obu przycisków. Po naciśnięciu pierwszego przycisku wartość zwrócona przez funkcję `millis()` jest zapisywana w pewnej zmiennej; po naciśnięciu drugiego przycisku wartość ponownie wywołanej funkcji `millis()` jest zapisywana w innej zmiennej. Funkcja niestandardowa `displayResult()` konwertuje wyznaczoną na tej podstawie wartość z milisekund na godziny, minuty i sekundy. I wreszcie czas pomiędzy naciśnięciem obu przycisków jest wyświetlany w oknie monitora portu szeregowego.

Sprzęt

Realizacja tego projektu będzie wymagała następujących elementów:

- płytki uniwersalnej;
- dwóch przycisków (S1 i S2);
- dwóch rezystorów 10 kΩ (R1 i R2);
- różnych przewodów;
- płytki Arduino i przewodu USB.

Schemat obwodu

Schemat obwodu pokazano na rysunku 8.30.

UWAGA *Ten sam obwód zostanie użyty w następnym projekcie, zatem nie warto go rozmontowywać po zakończeniu budowy urządzenia!*

Szkic

W środowisku Arduino IDE należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 31 — budowa stopera

unsigned long start, finished, elapsed;

void setup()
{
   Serial.begin(9600);
❶  pinMode(2, INPUT); // przycisk Start
   pinMode(3, INPUT); // przycisk Stop
   Serial.println("Naciśnij 1, aby uruchomić/wyzerować stoper; naciśnij
   ↪przycisk 2, aby wyświetlić zmierzony czas");
}
```

Rysunek 8.30. Schemat obwodu dla projektu nr 31

```
void displayResult()
{
   float h, m, s, ms;
   unsigned long over;

❷  elapsed = finished - start;

   h    = int(elapsed / 3600000);
   over = elapsed % 3600000;
   m    = int(over / 60000);
   over = over % 60000;
   s    = int(over / 1000);
   ms   = over % 1000;

   Serial.print("Zmierzony czas (milisekundy): ");
   Serial.println(elapsed);
   Serial.print("Zmierzony czas: ");
   Serial.print(h, 0);
   Serial.print(" h ");
   Serial.print(m, 0);
   Serial.print(" min ");
```

```
    Serial.print(s, 0);
    Serial.print(" s ");
    Serial.print(ms, 0);
    Serial.println(" ms");
    Serial.println();
}

void loop()
{
❸       if (digitalRead(2) == HIGH)
    {
        start = millis();
        delay(200); // zabezpieczenie przed drganiem styków
        Serial.println("Uruchomiono stoper...");
    }
❹       if (digitalRead(3) == HIGH)
    {
        finished = millis();
        delay(200); // zabezpieczenie przed drganiem styków
        displayResult();
    }
}
```

Podstawowe zasady działania tego stopera są proste. W wierszu ❶ ustawiono cyfrowe piny wejściowe dla przycisków uruchamiających i zatrzymujących stoper. Jeśli naciśnięto przycisk *Start*, w wierszu ❸ system Arduino zapisuje wartość zwróconą przez funkcję `millis()`, która będzie potrzebna do obliczenia zmierzonego czasu po naciśnięciu przycisku *Stop* w wierszu ❹. Naciśnięcie przycisku *Stop* powoduje obliczenie czasu pomiędzy oboma zdarzeniami w funkcji `displayResult()`❷ i wyświetlenie tego czasu w oknie monitora portu szeregowego.

W oknie monitora portu szeregowego powinny pojawić się dane podobne do tych z rysunku 8.31.

Rysunek 8.31. Dane wynikowe szkicu z projektu nr 31

Przerwania

Przerwanie (ang. *interrupt*) w systemie Arduino to sygnał umożliwiający wywołanie określonej funkcji w dowolnym czasie w trakcie wykonywania szkicu — na przykład po zmianie stanu cyfrowego pinu wejściowego lub po zdarzeniu wywołanym przez stoper. Przerwania doskonale nadają się do wywoływania funkcji przerywających normalne działanie szkicu, na przykład w reakcji na naciśnięcie przycisku.

Wywołanie przerwania powoduje tymczasowe wstrzymanie normalnego wykonywania programu (czyli kodu w ciele funkcji `void loop()`), wywołanie i wykonanie odpowiedniej funkcji oraz (po zakończeniu działania tej funkcji) powrót do głównej pętli i wznowienie jej wykonywania od punktu, w którym zostało przerwane.

Należy pamiętać, że funkcje przerwań powinny być możliwie krótkie i proste. Powinny jak najszybciej zwracać sterowanie do głównej pętli, a jeśli wykonują zadania, które mogą być realizowane także przez główną pętlę, należy pamiętać o możliwości tymczasowego nadpisania wyników działania tej pętli. Jeśli na przykład główna pętla szkicu cyklicznie wysyła słowo `Witaj` za pośrednictwem portu szeregowego i jeśli funkcja przerwania wysyła sekwencję znaków `---`, w monitorze portu szeregowego będą się pojawiały słowa: `W---itaj`, `Wi---taj`, `Wit---aj`, `Wita---j` lub `Witaj---`.

Płytka Arduino Uno obsługuje dwa przerwania dostępne za pośrednictwem pinów cyfrowych nr 2 i 3. Odpowiednio skonfigurowany system Arduino monitoruje napięcie na wymienionych pinach. W momencie zmiany tego napięcia w określony, zdefiniowany sposób (na przykład w wyniku naciśnięcia przycisku) jest wywoływane przerwanie powodujące uruchomienie odpowiedniej funkcji (na przykład wyświetlającej komunikat *Przestań mnie naciskać!*).

Tryby przerwań

Przerwanie może być wywoływane wskutek jednej z czterech zmian (tzw. trybów przerwań):

- `LOW` — na pinie przerwania nie ma napięcia.
- `CHANGE` — napięcie na pinie przerwania zmieniło się (było wyłączone i jest włączone lub było włączone i jest wyłączone).
- `RISING` — napięcie na pinie przerwania zmieniło się z wyłączonego na 5 V.
- `FALLING` — napięcie na pinie przerwania zmieniło się z 5 V na wyłączone.

Aby wykryć na przykład włączenie przycisku podłączonego do pinu przerwania, należy zastosować tryb `RISING`. A jeśli posesja jest zabezpieczona przewodem rozciągniętym wokół ogrodu (podłączonym do zasilania 5 V i pinu przerwania), należy zastosować tryb `FALLING`, by wykryć przerwanie tego przewodu.

UWAGA *Ewentualne wywołania funkcji `delay()` i `Serial.available()` nie będą działały w ciele funkcji wywołanej przez przerwanie.*

Konfiguracja przerwań

Aby skonfigurować przerwania, należy zastosować następujące wyrażenia w ciele funkcji `void setup()`:

```
attachInterrupt(0, function, mode);
attachInterrupt(1, function, mode);
```

W powyższych wywołaniach wartość 0 reprezentuje pin cyfrowy nr 2, wartość 1 reprezentuje pin cyfrowy nr 3, parametr `function` reprezentuje nazwę funkcji, która ma być wywołana przez przerwanie, a parametr `mode` jeden z czterech trybów przerwań.

Aktywowanie i dezaktywowanie przerwań

Zdarza się, że w pewnych miejscach szkicu należy wyłączyć obsługę przerwań. Przerwania można dezaktywować za pomocą następującego wywołania:

```
noInterrupts(); // dezaktywuje przerwania
```

Aby ponownie aktywować przerwania, należy użyć wywołania:

```
interrupts(); // ponownie aktywuje przerwania
```

Przerwania działają bardzo szybko i jako takie doskonale nadają się do realizacji najpilniejszych zadań lub na przykład do awaryjnego wyłączania urządzenia za pomocą odpowiedniego przycisku.

Projekt nr 32: stosowanie przerwań

Do prezentacji praktycznych zastosowań przerwań użyjemy obwodu z projektu nr 31. W tym przykładzie dioda LED na płytce Arduino będzie migała co 500 milisekund. W tym samym czasie system będzie monitorował oba piny przerwań. Naciśnięcie przycisku podłączonego do pinu przerwań nr 0 powoduje wyświetlenie w oknie monitora portu szeregowego wartości zwróconej przez funkcję `micros()`; naciśnięcie przycisku podłączonego do pinu przerwań nr 1 powoduje wyświetlenie wartości funkcji `millis()`.

Szkic

W środowisku Arduino IDE należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 32 — stosowanie przerwań

#define LED 13

void setup()
{
   Serial.begin(9600);
   pinMode(13, OUTPUT);
   attachInterrupt(0, displayMicros, RISING);
   attachInterrupt(1, displayMillis, RISING);
}

❶ void displayMicros()
{
   Serial.write("micros() = ");
   Serial.println(micros());
}

❷ void displayMillis()
{
   Serial.write("millis() = ");
   Serial.println(millis());
}

❸ void loop()
{
   digitalWrite(LED, HIGH);
   delay(500);
   digitalWrite(LED, LOW);
   delay(500);
}
```

Szkic powoduje miganie wbudowanej diody LED (w wierszu ❸ w ciele funkcji `void loop()`). Wywołanie przerwania nr 0 powoduje wywołanie funkcji `displayMicros()` ❶, a wywołanie przerwania nr 1 powoduje wywołanie funkcji `displayMillis()`❷. Po zakończeniu wykonywania jednej z tych funkcji szkic wznawia wykonywanie kodu funkcji `void loop()`.

Aby sprawdzić działanie tego szkicu, należy otworzyć okno monitora portu szeregowego i nacisnąć oba przyciski — program wyświetli wartości zwrócone przez funkcje `millis()` i `micros()` (patrz rysunek 8.32).

Rysunek 8.32. Dane wynikowe szkicu z projektu nr 32

Co dalej?

W tym rozdziale wprowadziłem dużo narzędzi i opcji, które będą stosowane w dalszych projektach i które pozwolą usprawnić projekty realizowane przez Ciebie. W kolejnych rozdziałach omówię dodatkowe moduły dla platformy Arduino i pokażę, jak wykorzystać moduł kart pamięci microSD w innych aplikacjach rejestrujących dane.

9

Klawiatury numeryczne

W tym rozdziale:

- dowiesz się, jak łączyć klawiatury numeryczne z płytką Arduino;
- nauczysz się odczytywać wartości z klawiatury numerycznej w kodzie szkicu;
- rozszerzysz systemy podejmowania decyzji za pomocą konstrukcji `switch-case`;
- zbudujesz zamek lub przełącznik zabezpieczony kodem PIN.

Stosowanie klawiatury numerycznej

Ponieważ realizowane projekty są coraz bardziej rozbudowane, warto opanować umiejętność pobierania od użytkowników danych numerycznych także w sytuacji, gdy płytka Arduino nie jest połączona z komputerem PC — jeśli na przykład urządzenie ma być włączane lub wyłączane poprzez wpisanie jakiejś tajnej liczby. Jednym z możliwych rozwiązań jest podłączenie co najmniej dziesięciu przycisków do różnych cyfrowych pinów wejściowych (dla cyfr od 0 do 9), jednak dużo prostszym wyjściem jest użycie klawiatury numerycznej podobnej do tej z rysunku 9.1.

Rysunek 9.1. Klawiatura numeryczna

Jedną z zalet stosowania klawiatury numerycznej jest wykorzystanie zaledwie 7 pinów dla 12 aktywnych przycisków oraz możliwość użycia gotowej biblioteki dla platformy Arduino, która eliminuje konieczność stosowania rezystorów zabezpieczających układ przed efektem drgania styków (jak w rozdziale 4.). Biblioteka *Keypad* jest dostępna do pobrania pod adresem *http://playground.arduino.cc/uploads/Code/Keypad.zip*.

Łączenie klawiatury numerycznej

Fizyczne łączenie klawiatury numerycznej z płytką Arduino jest proste. Na spodzie klawiatury numerycznej znajduje się siedem pinów (patrz rysunek 9.2).

Rysunek 9.2. Piny klawiatury numerycznej

Piny klawiatury numerycznej oznaczono numerami od 1 do 7 (od lewej do prawej strony). We wszystkich opisanych w tej książce projektach obejmujących klawiaturę numeryczną będzie stosowany schemat połączeń taki jak w tabeli 9.1.

Tabela 9.1. Połączenia klawiatury numerycznej z płytką Arduino

Numer pinu klawiatury numerycznej	Pin na płytce Arduino
1	Pin cyfrowy nr 5
2	Pin cyfrowy nr 4
3	Pin cyfrowy nr 3
4	Pin cyfrowy nr 2
5	Pin cyfrowy nr 8
6	Pin cyfrowy nr 7
7	Pin cyfrowy nr 6

Programowanie obsługi klawiatury numerycznej

Szkic korzystający z klawiatury numerycznej musi zawierać pewne wiersze niezbędne do obsługi tego urządzenia (patrz poniższy przykład). Potrzebny kod rozpoczyna się w wierszu ❶ i kończy w wierszu ❺.

Przed przystąpieniem do dalszych kroków warto przetestować klawiaturę numeryczną, wpisując i wysyłając na płytkę Arduino szkic z listingu 9.1.

Listing 9.1. Szkic demonstrujący działanie klawiatury numerycznej

```
// Listing 9.1

❶ // Początek niezbędnego kodu

#include "Keypad.h"
const byte ROWS = 4; // ustawia układ złożony z czterech wierszy
const byte COLS = 3; // ustawia układ złożony z trzech kolumn
char keys[ROWS][COLS] =
 {{'1','2','3'},
   {'4','5','6'},
   {'7','8','9'},
❷   {'*','0','#'}};
❸   byte rowPins[ROWS] = {5, 4, 3, 2};
❹   byte colPins[COLS] = {8, 7, 6};
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

❺ // Koniec niezbędnego kodu

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   char key = keypad.getKey();
   if (key != NO_KEY)
   {
      Serial.print(key);
   }
}
```

W wierszu ❷ po raz pierwszy zastosowano zmienną typu char reprezentującą pojedynczy znak, na przykład literę, cyfrę lub symbol, generowany przez klawiaturę komputerową. W tym przypadku tablica typu char zawiera cyfry i symbole dostępne na klawiaturze numerycznej. Wiersze kodu ❸ i ❹ określają, które piny na płytce Arduino będą używane przez klawiaturę. W razie potrzeby za pomocą tych dwóch wierszy i schematu opisanego w tabeli 9.1 można zmienić cyfrowe piny wejściowe.

Testowanie szkicu

Po wysłaniu szkicu na płytkę Arduino należy otworzyć okno monitora portu szeregowego i nacisnąć kilka klawiszy klawiatury numerycznej. Znaki reprezentowane przez naciśnięte klawisze zostaną wyświetlone w oknie monitora portu szeregowego (patrz rysunek 9.3).

Rysunek 9.3. Efekt naciskania klawiszy klawiatury numerycznej

Podejmowanie decyzji za pomocą konstrukcji switch-case

Jeśli w kodzie szkicu należy porównywać co najmniej dwie zmienne, w wielu przypadkach użycie konstrukcji `switch-case` jest prostszym rozwiązaniem niż stosowanie wielu wyrażeń `if-then`, ponieważ wyrażenia `switch-case` umożliwiają definiowanie dowolnej liczby porównań i wykonywanie odpowiedniego kodu dopiero po znalezieniu prawdziwego warunku. Jeśli na przykład zmienna całkowitoliczbowa `xx` może mieć wartość 1, 2 lub 3 i jeśli chcemy wykonać różne fragmenty kodu zależnie od wartości tej zmiennej, możemy użyć następującej konstrukcji zamiast wielu wyrażeń `if-then`:

```
switch(xx)
{
   case 1:
      // wykonuje jakąś operację, jeśli xx ma wartość 1
      break; // kończy wyrażenie switch i przechodzi do dalszej części szkicu
   case 2:
      // wykonuje jakąś operację, jeśli xx ma wartość 2
      break;
   case 3:
      // wykonuje jakąś operację, jeśli xx ma wartość 3
      break;
   default:
      // wykonuje jakąś operację, jeśli xx ma wartość różną od 1, 2 i 3
      // element default jest opcjonalny
}
```

Opcjonalna sekcja `default:` na końcu tego kodu umożliwia zdefiniowanie kodu, który będzie wykonywany w razie braku pasującej wartości w sekcjach `case` wyrażenia `switch`.

Projekt nr 33: tworzenie zamka sterowanego klawiaturą numeryczną

W tym projekcie zbudujemy pierwszy element zamka sterowanego klawiaturą numeryczną. W projekcie zostanie użyty podstawowy kod konfigurujący klawiaturę numeryczną z listingu 9.1 oraz dodatkowy, tajny kod, który użytkownik będzie musiał wpisać na klawiaturze numerycznej. W oknie monitora portu szeregowego zostanie wyświetlona informacja, czy kod wpisany na klawiaturze numerycznej jest poprawny.

Szkic dla tego projektu wywołuje różne funkcje w zależności od tego, czy tajny, sześciocyfrowy kod jest poprawny. Tajny kod jest co prawda przechowywany w kodzie szkicu, ale nie jest wyświetlany użytkownikowi. Aby aktywować i dezaktywować zamek, użytkownik musi nacisnąć klawisz *, wpisać tajny numer i nacisnąć klawisz #.

Szkic

W środowisku Arduino IDE należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 33 — tworzenie zamka sterowanego klawiaturą numeryczną

// Początek niezbędnego kodu

#include "Keypad.h"
const byte ROWS = 4; // ustawia układ złożony z czterech wierszy
const byte COLS = 3; // ustawia układ złożony z trzech kolumn
char keys[ROWS][COLS] =
 {{'1','2','3'},
   {'4','5','6'},
   {'7','8','9'},
   {'*','0','#'}};
byte rowPins[ROWS] = {5, 4, 3, 2};
byte colPins[COLS] = {8, 7, 6};
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );

// Koniec niezbędnego kodu

❶ char PIN[6]={'1','2','3','4','5','6'}; // tajna liczba
char attempt[6]={0,0,0,0,0,0};
int z=0;
```

```
void setup()
{
   Serial.begin(9600);
}

void correctPIN() // funkcja wywoływana, jeśli wpisano poprawny kod PIN
{
   Serial.println("Wpisano prawidłowy kod PIN...");
}

void incorrectPIN() // funkcja wywoływana, jeśli kod PIN jest nieprawidłowy
{
   Serial.println("Wpisano nieprawidłowy kod PIN!");
}

void checkPIN()
{
   int correct=0;
   int i;
❷  for ( i = 0; i < 6 ; i++ )
   {
      if (attempt[i]==PIN[i])
      {
         correct++;
      }
   }
   if (correct==6)
   {
❸  correctPIN();
   } else
   {
❹  incorrectPIN();
   }
   for (int zz=0; zz<6; zz++) // usuwa kod PIN wpisany w poprzedniej próbie
   {
      attempt[zz]=0;
   }
}

void readKeypad()
{
   char key = keypad.getKey();
   if (key != NO_KEY)
   {
❺   switch(key)
      {
         case '*':
            z=0;
            break;
         case '#':
            delay(100); // eliminuje skutki zjawiska drgania styków
```

```
            checkPIN();
            break;
         default:
            attempt[z]=key;
            z++;
      }
   }
}

void loop()
{
❻  readKeypad();
}
```

Działanie szkicu

Po standardowych wierszach konfigurujących klawiaturę numeryczną (takich jak na listingu 9.1) szkic rozpoczyna „nasłuchiwanie" tej klawiatury poprzez uruchomienie funkcji `readKeypad()` w wierszu ❻. Naciśnięcie klawisza powoduje sprawdzenie przekazanej wartości w ramach wyrażenia `switch-case` w wierszu ❺. Wartości naciśniętych klawiszy klawiatury numerycznej są przechowywane w tablicy `attempt[]`. Naciśnięcie przez użytkownika klawisza # powoduje wywołanie funkcji `checkPin()`.

W wierszu ❷ wartości naciśniętych klawiszy są porównywane z kodem zapisanym w tablicy `PIN[]` ❶, czyli tajnym numerem PIN. Jeśli użytkownik wpisał prawidłową sekwencję cyfr, jest wywoływana funkcja `correctPin()` w wierszu ❸ (funkcję można uzupełnić o własny kod wykonywany po podaniu prawidłowego numeru PIN); jeśli wpisany numer PIN jest nieprawidłowy, w wierszu ❹ jest wywoływana funkcja `incorrectPin()`. I wreszcie po sprawdzeniu poprawności wpisanego kodu PIN sekwencja cyfr jest usuwana z pamięci, a kod szkicu jest gotowy na następny test.

Testowanie szkicu

Po wysłaniu szkicu na platformę Arduino należy otworzyć okno monitora portu szeregowego, nacisnąć klawisz gwiazdki (*) na klawiaturze numerycznej, wpisać tajny numer PIN i nacisnąć klawisz krzyżyka (#). Warto spróbować wpisać zarówno prawidłowy, jak i nieprawidłowy numer PIN. Wynik tego testu powinien przypominać dane wyjściowe pokazane na rysunku 9.4.

Opisany przykład stanowi doskonały punkt wyjścia dla własnych urządzeń aktywowanych za pomocą numerów PIN, jak zamki, alarmy, lub dowolnej innej konstrukcji wymyślonej przez Ciebie. Wystarczy zastąpić kod funkcji `correctPIN()` i `incorrectPIN()` rozwiązaniami odpowiadającymi za właściwe reagowanie na wpisanie odpowiednio prawidłowej i nieprawidłowej sekwencji cyfr.

Rysunek 9.4. Wynik wpisania prawidłowych i nieprawidłowych numerów PIN

Co dalej?

W tym rozdziale wprowadziłem kolejną technikę uzyskiwania danych wejściowych przez system Arduino. Wiedza nabyta podczas lektury tego rozdziału stanowi cenny punkt wyjścia dla dalszych rozwiązań obejmujących sterowanie szkicem za pośrednictwem klawiatury numerycznej oraz kontrolowanie dostępu do urządzeń za pomocą zamka szyfrowego. Wyjaśniłem też działanie bardzo przydatnej konstrukcji `switch-case`. W następnym rozdziale wprowadzę jeszcze inną formę uzyskiwania danych wejściowych — ekran dotykowy.

10

Odczytywanie danych wejściowych użytkownika za pośrednictwem ekranów dotykowych

W tym rozdziale:

- nauczysz się łączyć oporowy ekran dotykowy z platformą Arduino;
- dowiesz się, jakie wartości może zwracać ekran dotykowy;
- zbudujesz prosty przełącznik dotykowy (włączony/wyłączony);
- opracujesz dotykowy włącznik z regulacją jasności.

Z ekranami dotykowymi mamy do czynienia praktycznie codziennie — ekrany tego typu stosuje się w smartfonach, tabletach czy przenośnych konsolach do gier. Dlaczego więc nie użyć ekranu dotykowego do uzyskiwania danych wejściowych od użytkownika?

Ekrany dotykowe

Ekrany dotykowe mogą być dość drogie, jednak w tym rozdziale zostanie wykorzystany tani model dostępny w sklepie SparkFun (nr towaru: LCD-08977 i BOB-09170) zaprojektowany z myślą o konsoli do gier Nintendo DS.

Wymiary tego ekranu dotykowego wynoszą około 2 na 2,75 cala. Przykład montażu ekranu na płytce uniwersalnej pokazano na rysunku 10.1.

Rysunek 10.1. Ekran dotykowy zamontowany na płytce uniwersalnej

Warto zwrócić uwagę na poziomy przewód taśmowy podłączony do niewielkiego obwodu w prawym górnym rogu urządzenia (zakreślony fragment). Wspomniany układ umożliwia łączenie ekranu dotykowego z platformą Arduino i płytką uniwersalną (na rysunku 10.2 pokazano zbliżenie tego układu).

Rysunek 10.2. Płytka umożliwiająca łączenie ekranu dotykowego

Łączenie ekranu dotykowego

Płytkę ekranu dotykowego należy połączyć z płytką Arduino zgodnie ze schematem opisanym w tabeli 10.1.

Tabela 10.1. Połączenia pinów płytki ekranu dotykowego z pinami na płytce Arduino

Pin płytki ekranu dotykowego	Pin na płytce Arduino
Y1	A0
X1	A1
Y2	A2
X2	A3

Projekt nr 34: adresowanie obszarów na ekranie dotykowym

Ekran dotykowy składa się z dwóch przezroczystych powłok — górna warstwa jest wykonana z folii, dolna warstwa ze szkła. Pierwsza powłoka odpowiada za identyfikację punktu dotknięcia w osi X, druga odpowiada za lokalizację tego punktu w osi Y. Oporność obu powłok zależy od miejsca ewentualnego dotknięcia powierzchni ekranu, zatem pomiar napięcia umożliwia określenie punktu dotknięcia zarówno w poziomie (oś X), jak i w pionie (oś Y).

W tym projekcie platforma Arduino zostanie użyta do rejestrowania punktów dotknięcia ekranu i konwersji tych danych na liczby całkowite niezbędne do wskazania odpowiednich obszarów ekranu.

Sprzęt

Realizacja projektu będzie wymagała następującego sprzętu:

- ekranu dotykowego z płytką połączeniową;
- trimpota 10 kΩ;
- znakowego modułu LCD 16×2;
- różnych przewodów;
- płytki uniwersalnej;
- płytki Arduino i przewodu USB.

Ekran dotykowy należy połączyć z płytką Arduino zgodnie ze schematem opisanym w tabeli 10.1, natomiast moduł LCD należy podłączyć zgodnie ze schematem pokazanym na rysunku 7.2 w rozdziale 7.

Szkic

Należy teraz wpisać i wysłać na płytkę Arduino następujący szkic. Najważniejsze elementy tego szkicu oznaczyłem odpowiednimi komentarzami:

```
// Projekt nr 34 — adresowanie obszarów na ekranie dotykowym

#include <LiquidCrystal.h>
LiquidCrystal lcd(4,5,6,7,8,9);

int x,y = 0;

void setup()
{
   lcd.begin(16,2);
   lcd.clear();
}

❶ int readX() // zwraca wartość dla osi X ekranu dotykowego
{
   int xr=0;
   pinMode(A0, INPUT);
   pinMode(A1, OUTPUT);
   pinMode(A2, INPUT);
   pinMode(A3, OUTPUT);
   digitalWrite(A1, LOW);  // ustawia GND na pinie A1
   digitalWrite(A3, HIGH); // ustawia 5 V na pinie A3
   delay(5);
   xr=analogRead(0);       // zapisuje wartość dla osi X
   return xr;
}

❷ int readY() // zwraca wartość dla osi Y ekranu dotykowego
{
   int yr=0;
   pinMode(A0, OUTPUT);    // A0
   pinMode(A1, INPUT);     // A1
   pinMode(A2, OUTPUT);    // A2
   pinMode(A3, INPUT);     // A3
   digitalWrite(14, LOW);  // ustawia GND na pinie A0
   digitalWrite(16, HIGH); // ustawia 5 V na pinie A2
   delay(5);
   yr=analogRead(1); // zapisuje wartość dla osi Y
   return yr;
}

void loop()
{
   lcd.setCursor(0,0);
❸  lcd.print(" x = ");
   x=readX();
   lcd.print(x);
   y=readY();
   lcd.setCursor(0,1);
```

```
❹   lcd.print(" y = ");
    lcd.print(y);
    delay (200);
}
```

Funkcje `readX()` i `readY()` w wierszach ❶ i ❷ odczytują napięcie prądu elektrycznego z obu warstw oporowego ekranu dotykowego, mierzą to napięcie za pomocą funkcji `analogRead()` i zwracają uzyskaną wartość. Szkic błyskawicznie wykonuje te funkcje niemal bezpośrednio po sobie, aby wyznaczyć dotykany obszar ekranu w czasie rzeczywistym i wyświetlić punkt dotknięcia na ekranie LCD (w wierszach ❸ i ❹). (Wywołanie `delay(5)` w obu wspomnianych funkcjach jest konieczne, ponieważ piny wejściowe i wyjściowe potrzebują czasu na zmianę swojego stanu).

Testowanie szkicu

Aby przetestować ten szkic, należy obserwować moduł LCD w czasie dotykania ekranu dotykowego i analizować zmiany wartości X i Y w miarę przesuwania palca na powierzchni ekranu. Warto też zwrócić uwagę na wartości wyświetlane w czasie, gdy ekran dotykowy nie jest dotykany (patrz rysunek 10.3).

Rysunek 10.3. Wartości wyświetlane w czasie, gdy ekran nie jest dotykany

Analiza tych wartości jest o tyle ważna, że właśnie na ich podstawie będzie można wykrywać w kodzie szkicu, kiedy ekran nie jest dotykany.

Odwzorowywanie punktów dotknięcia ekranu

Dotykając narożników ekranu dotykowego i rejestrując zwracane wartości, można przygotować swoistą mapę tego ekranu (patrz rysunek 10.4). Na mapę należy nanieść współrzędne wszystkich czterech narożników. Po określeniu tych wartości można przystąpić do dzielenia mapy ekranu dotykowego na mniejsze obszary używane jako powierzchnie sterowania.

Po przygotowaniu mapy ekranu dotykowego można ten ekran podzielić na mniejsze obszary, a następnie używać funkcji `readX()` i `readY()` do identyfikacji aktualnie wskazywanego (dotykanego) obszaru. Decyzję o wyborze działania w zależności od dotykanego obszaru można podejmować za pomocą wyrażeń `if-then` (patrz następny projekt).

Rysunek 10.4. Mapa ekranu dotykowego

Projekt nr 35: budowa dwustanowego przełącznika dotykowego

W tym projekcie opracowana wcześniej mapa ekranu dotykowego zostanie użyta do budowy przełącznika dwustanowego (włączony/wyłączony). Ekran dotykowy należy podzielić w pionie na dwie połowy (patrz rysunek 10.5). Lewa strona będzie służyła do włączania, a prawa do wyłączania.

Rysunek 10.5. Mapa ekranu dotykowego pełniącego funkcję włącznika

System Arduino określi, która strefa ekranu dotykowego została dotknięta przez użytkownika, porównując zarejestrowane współrzędne dotknięcia z granicami obu obszarów zajmujących po połowie tego ekranu. Po określeniu dotkniętego

obszaru można użyć pinu cyfrowego do wysłania sygnału włączenia lub wyłączenia, jednak ten szkic ograniczy się tylko do wyświetlania naciśniętego obszaru (włączania lub wyłączania) w oknie monitora portu szeregowego.

Szkic

W środowisku IDE należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 35 — budowa dwustanowego przełącznika dotykowego

int x,y = 0;

void setup()
{
   Serial.begin(9600);
   pinMode(10, OUTPUT);
}

void switchOn()
{
   digitalWrite(10, HIGH);
   Serial.print("Włączony w punkcie X = ");
   Serial.print(x);
   Serial.print(" Y = ");
   Serial.println(y);
   delay(200);
}

void switchOff()
{
   digitalWrite(10, LOW);
   Serial.print("Wyłączony w punkcie X = ");
   Serial.print(x);
   Serial.print(" Y = ");
   Serial.println(y);
   delay(200);
}

int readX() // zwraca wartość dla osi X ekranu dotykowego
{
   int xr=0;
   pinMode(A0, INPUT);
   pinMode(A1, OUTPUT);
   pinMode(A2, INPUT);
   pinMode(A3, OUTPUT);
   digitalWrite(A1, LOW);  // ustawia GND na pinie A1
   digitalWrite(A3, HIGH); // ustawia 5 V na pinie A3
   delay(5);
   xr=analogRead(0);
   return xr;
```

```
}

int readY() // zwraca wartość dla osi Y ekranu dotykowego
{
   int yr=0;
   pinMode(A0, OUTPUT);
   pinMode(A1, INPUT);
   pinMode(A2, OUTPUT);
   pinMode(A3, INPUT);
   digitalWrite(A0, LOW);  // ustawia GND na pinie A0
   digitalWrite(A2, HIGH); // ustawia 5 V na pinie A2
   delay(5);
   yr=analogRead(1);
   return yr;
}

void loop()
{
   x=readX();
   y=readY();

❶  // sprawdza stan włączony

   if (x<=900 && x>=500)
   {
      switchOn();
   }

❷  // sprawdza stan wyłączony

   if (x<500 && x>=100)
   {
      switchOff();
   }
}
```

Działanie szkicu

Dwa wyrażenia `if` w ciele funkcji `void loop()` sprawdzają, czy została dotknięta lewa i prawa strona ekranu dotykowego. Jeśli użytkownik dotknął lewej strony, dotknięcie jest interpretowane jako włączenie przełącznika ❶. Jeśli użytkownik dotknął prawej strony, dotknięcie jest interpretowane jako wyłączenie przełącznika ❷.

UWAGA *Ponieważ ekran dotykowy podzielono tylko w pionie, współrzędne pionowe (na osi Y) są ignorowane. Gdyby ekran dotykowy podzielono dodatkowo w poziomie, należałoby sprawdzać także oś Y (patrz projekt nr 36 w dalszej części tego rozdziału).*

Testowanie szkicu

Szkic powinien wyświetlić w oknie monitora portu szeregowego dane wynikowe podobne do tych z rysunku 10.6. Stan przełącznika i współrzędne dotknięcia są wyświetlane po każdym dotknięciu ekranu.

Rysunek 10.6. Dane wynikowe wygenerowane przez szkic dla projektu nr 35

Projekt nr 36: budowa przełącznika dotykowego podzielonego na trzy obszary

W tym projekcie zbudujemy przełącznik dotykowy podzielony na trzy obszary. Za pomocą przełącznika będzie można nie tylko włączać i wyłączać diodę LED podłączoną do pinu cyfrowego nr 3, ale też zmieniać jasność świecenia tej diody (przy użyciu opisanej w rozdziale 3. techniki modulacji szerokości impulsu).

Mapa ekranu dotykowego

Mapę ekranu dotykowego pokazano na rysunku 10.7.

Rysunek 10.7. Mapa ekranu dotykowego dla projektu nr 36

Mapę ekranu dotykowego podzielono na strefy wyłączania i włączania oraz dodatkową strefę sterowania jasnością świecenia diody. Wartości zwracane przez ekran dotykowy są analizowane w celu określenia, która część tego ekranu została dotknięta, aby na tej podstawie wybrać właściwe działania.

Szkic

W środowisku IDE należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 36 — budowa przełącznika dotykowego podzielonego na trzy obszary

int x,y = 0;

void setup()
{
   pinMode(3, OUTPUT);
}

void switchOn()
{
   digitalWrite(3, HIGH);
   delay(200);
}

void switchOff()
{
   digitalWrite(3, LOW);
   delay(200);
}

void setBrightness()
{
   int xx, bright;
   float br;
   xx=x-100;
❶  br=(800-xx)/255;
   bright=int(br);
   analogWrite(3, bright);
}

int readX() // zwraca wartość dla osi X ekranu dotykowego
{
   int xr=0;
   pinMode(A0, INPUT);
   pinMode(A1, OUTPUT);
   pinMode(A2, INPUT);
   pinMode(A3, OUTPUT);
   digitalWrite(A1, LOW);  // ustawia GND na pinie A1
   digitalWrite(A3, HIGH); // ustawia 5 V na pinie A3
   delay(5);
```

```
   xr=analogRead(0);
   return xr;
}

int readY() // zwraca wartość dla osi Y ekranu dotykowego
{
   int yr=0;
   pinMode(A0, OUTPUT);    // A0
   pinMode(A1, INPUT);     // A1
   pinMode(A2, OUTPUT);    // A2
   pinMode(A3, INPUT);     // A3
   digitalWrite(A0, LOW);  // ustawia GND na pinie A0
   digitalWrite(A2, HIGH); // ustawia 5 V na pinie A2
   delay(5);
   yr=analogRead(1);
   return yr;
}

void loop()
{
   x=readX();
   y=readY();

   // sprawdza stan włączony

❷   if (x<=500 && x>=100 && y>= 150 && y<375)
   {
      switchOn();
   }

   // sprawdza stan wyłączony

❸  if (x>500 && x<=900 && y>= 150 && y<375)
   {
      switchOff();
   }

   // sprawdza, czy dotknięto obszaru sterowania jasnością

❹  if (y>=375 && y<=900)
   {
      setBrightness();
   }
}
```

Działanie szkicu

Tak jak w szkicu dla mapy podzielonej na dwie strefy, w tym szkicu są sprawdzane ewentualne dotknięcia sekcji włączania i wyłączania (teraz nieco mniejszych) w wierszach ❷ i ❸ i dotknięcia poziomego paska regulacji jasności w wierszu ❹.

Jeśli ekran jest dotknięty w obszarze regulacji jasności, położenie punktu dotknięcia w poziomie (na osi X) jest konwertowane na względną wartość modulacji szerokości impulsu (PWM) w wierszu ❶, a jasność świecenia diody LED jest odpowiednio dostosowywana za pomocą funkcji `setBrightness()`.

Za pomocą różnych kombinacji opisanych funkcji można tworzyć dowolną liczbę przełączników lub suwaków na powierzchni niedrogiego ekranu dotykowego.

Co dalej?

W tym rozdziale wprowadziłem jeszcze jedną metodę otrzymywania danych wejściowych użytkownika i sterowania systemem Arduino. W następnym rozdziale skoncentruję się na samej płytce Arduino — omówię dostępne wersje i opiszę możliwy sposób budowy własnej wersji na płytce uniwersalnej.

11

Rodzina produktów Arduino

W tym rozdziale:

- nauczysz się budować własne układy Arduino na płytce uniwersalnej;
- poznasz możliwości i zalety dostępnych na rynku płytek zgodnych z platformą Arduino;
- dowiesz się, czym jest sprzęt open source.

Podzielimy projekt płytki Arduino na kilka grup elementów, a następnie zbudujemy na płytce uniwersalnej własny obwód zgodny z tym standardem. Budując własny obwód, można zaoszczędzić pieniądze, szczególnie jeśli ten obwód jest używany do przygotowywania zmieniających się projektów i prototypów. W tym rozdziale wprowadzę też kilka nowych komponentów elektronicznych. Pokażę również, jak wysyłać szkice na własnoręcznie budowane płytki Arduino bez konieczności stosowania dodatkowego sprzętu. I wreszcie omówię najbardziej popularne alternatywy dla płytki Arduino Uno i przeanalizuję różnice dzielące te urządzenia.

Projekt nr 37: budowa własnej platformy Arduino na płytce uniwersalnej

Wraz ze wzrostem złożoności i liczby realizowanych projektów i eksperymentów koszt zakupu samych płytek Arduino może łatwo wymknąć się spod kontroli, szczególnie jeśli konstruktor preferuje pracę nad wieloma projektami jednocześnie. W takim przypadku tańszym i prostszym rozwiązaniem jest łączenie obwodów budowanych na potrzeby projektu z własnoręcznie przygotowaną platformą Arduino na płytce uniwersalnej, którą można łatwo dostosowywać na potrzeby tworzonych urządzeń. Odtworzenie podstawowego układu platformy Arduino na płytce uniwersalnej nie powinno kosztować więcej niż 50 zł (co więcej, tak zbudowany układ można stosować wiele razy, o ile nie będzie poddawany zbyt dużemu obciążeniu). Własna platforma Arduino dodatkowo upraszcza realizację projektów wymagających złożonych obwodów, ponieważ eliminuje konieczność łączenia płytki Arduino z płytką uniwersalną za pomocą dużej liczby przewodów.

Sprzęt

Budowa minimalistycznej wersji platformy Arduino wymaga użycia następujących urządzeń:

- płytki uniwersalnej;
- różnych przewodów;
- liniowego regulatora napięcia 7805;
- rezonatora kwarcowego 16 MHz (na przykład produkt nr 16C8140 w sklepie Newark);
- mikrokontrolera ATmega328P-PU z Arduino z programem rozruchowym Arduino;
- kondensatora elektrolitycznego 1 μF, 25 V (C1);
- kondensatora elektrolitycznego 100 μF, 25 V (C2);
- dwóch kondensatorów ceramicznych 22 pF, 50 V (C3 i C4);
- kondensatora ceramicznego 100 nF, 50 V (C5);
- dwóch rezystorów 560 kΩ (R1 i R2);
- rezystora 10 kΩ (R3);
- dwóch dowolnych diod LED (LED1 i LED2);
- przycisku (S1);
- złącza sześciopinowego;
- złącza baterii typu PP3;
- baterii 9 V typu PP3.

Niektóre spośród wymienionych części być może są dla Ciebie zupełną nowością. W poniższych podpunktach wyjaśnię działanie wszystkich nowo wprowadzonych elementów i przedstawię ich symbole stosowane na schematach obwodów.

Liniowy regulator napięcia 7805

Liniowy regulator napięcia to prosty obwód konwertujący jedną wartość napięcia na inną. Na liście potrzebnych części umieściłem regulator typu 7805, który konwertuje napięcie z przedziału od 7 do 30 woltów na stałe napięcie 5 woltów i natężenie nie wyższe niż 1 amper, czyli wprost idealne parametry do zasilania płytki uniwersalnej pełniącej funkcję platformy Arduino. Przykład regulatora napięcia typu 7805 (nazwanego TO-220, od fizycznego kształtu) pokazano na rysunku 11.1.

Rysunek 11.1. Liniowy regulator napięcia 7805

Na rysunku 11.2 pokazano symbol regulatora napięcia 7805 stosowany na schematach obwodów. Jeśli patrzymy na regulator 7805 od strony etykiety, pin po lewej stronie służy do przekazywania napięcia wejściowego, pin środkowy należy połączyć z uziemieniem, a pin po prawej stronie służy do przekazywania napięcia wyjściowego (5 V). Metalowa blaszka nad regulatorem ma wywiercony otwór umożliwiający podłączenie większego metalowego elementu (tzw. radiatora). Radiatory stosuje się w obwodach zasilanych prądem o natężeniu zbliżonym do maksymalnego poziomu 1 A, ponieważ w takich warunkach regulator 7805 dość mocno się nagrzewa. Blaszka jest połączona z uziemieniem (GND). W tym przykładzie będzie potrzebny jeden regulator 7805.

Rysunek 11.2. Symbol regulatora napięcia 7805 stosowany na schematach obwodów

Rezonator kwarcowy 16 MHz

Rezonator kwarcowy (ang. *crystal oscillator*) służy do generowania sygnału elektrycznego o precyzyjnie określonej częstotliwości. W tym przypadku częstotliwość wynosi 16 MHz. Rezonator kwarcowy, który zostanie użyty w tym projekcie, pokazano na rysunku 11.3.

Rysunek 11.3. Rezonator kwarcowy

Warto porównać ten komponent z rezonatorem kwarcowym na płytce Arduino. Oba urządzenia powinny mieć identyczny kształt i te same wymiary.

Rezonatory kwarcowe nie są spolaryzowane. Symbol tego komponentu stosowany na schematach obwodów pokazano na rysunku 11.4.

Rysunek 11.4. Symbol rezonatora kwarcowego

Rezonator kwarcowy bezpośrednio wpływa na szybkość działania mikrokontrolera. Stosowany w tym obwodzie mikrokontroler jest taktowany z częstotliwością 16 MHz, co oznacza, że może wykonać 16 milionów rozkazów procesora w ciągu sekundy. Nie należy jednak zakładać, że równie szybko będzie wykonywany pojedynczy wiersz szkicu, ponieważ każdy wiersz kodu wymaga wykonania wielu rozkazów procesora.

Mikrokontroler Atmel ATmega328P-PU

Mikrokontroler to miniaturowy komputer obejmujący procesor (odpowiedzialny za wykonywanie rozkazów), różne rodzaje pamięci (używane do przechowywania danych i rozkazów przekazywanych przez szkic) oraz komponenty niezbędne do wysyłania i odbierania danych. Jak wspomniałem w rozdziale 2., właśnie mikrokontroler jest mózgiem platformy Arduino. Przykład mikrokontrolera ATmega328P pokazano na rysunku 11.5. Po ustawieniu tego

Rysunek 11.5. Mikrokontroler ATmega328P

układu scalonego tak jak na zdjęciu pin numer 1 znajduje się w lewym dolnym rogu układu (jest oznaczony małą kropką).

Na rysunku 11.6 pokazano symbol tego mikrokontrolera stosowany na schematach obwodów.

Rysunek 11.6. Symbol mikrokontrolera stosowany na schematach obwodów

Nie wszystkie mikrokontrolery zawierają *program rozruchowy* systemu Arduino, czyli oprogramowanie niezbędne do interpretacji szkiców pisanych dla tej platformy. Przy zakupie mikrokontrolera na potrzeby samodzielnie konstruowanej płytki Arduino koniecznie należy wybrać model zawierający program rozruchowy. Odpowiednie mikrokontrolery można bez trudu znaleźć w sklepach oferujących oryginalne płytki Arduino, w tym w sklepach adafruit, Freetronics i SparkFun.

Schemat obwodu

Schemat obwodu dla tego projektu pokazano na rysunku 11.7.

Rysunek 11.7. Schemat obwodu platformy Arduino na płytce uniwersalnej

Schemat składa się z dwóch sekcji. Pierwsza sekcja, po lewej stronie, odpowiada za zasilanie i ograniczanie napięcia do 5 V. Włączone zasilanie jest sygnalizowane przez świecącą diodę LED. Druga sekcja, po prawej stronie, składa się z mikrokontrolera, przycisku *RESET*, programowalnych pinów i jeszcze

jednej diody LED. Druga dioda LED jest połączona z pinem mikrokontrolera ATmega328P używanym jako pin nr 13 platformy Arduino. Własną platformę Arduino należy zbudować na podstawie tego schematu. Nie wolno zapominać o doprowadzeniu przewodów do złącza sześciopinowego (patrz rysunek 11.8) reprezentowanego przez sześć okręgów na dole schematu. Złącze zostanie użyte w dalszej części tego rozdziału do wysyłania szkicu na własnoręcznie zbudowaną płytkę Arduino.

Rysunek 11.8. Złącze sześciopinowe

Obwód będzie zasilany za pomocą baterii 9 V przy użyciu odpowiedniego złącza baterii (patrz rysunek 11.9). Czerwony przewód złącza baterii należy połączyć ze stroną dodatnią obwodu (+); czarny przewód tego złącza należy połączyć ze stroną ujemną (–).

Rysunek 11.9. Bateria 9 V ze złączem

Identyfikacja pinów platformy Arduino

Gdzie podziały się wszystkie piny dostępne na oryginalnej płytce Arduino? Okazuje się, że wszystkie piny analogowe, cyfrowe i inne są dostępne także w wersji platformy Arduino zbudowanej na płytce uniwersalnej — wystarczy podłączenie do odpowiednich nóżek mikrokontrolera.

W prezentowanym modelu platformy Arduino rezystor R2 i dioda LED2 są połączone z pinem cyfrowym nr 13. W tabeli 11.1 wymieniono piny platformy Arduino (w pierwszej kolumnie) i odpowiednie piny mikrokontrolera ATmega328P (w prawej kolumnie).

Tabela 11.1. Piny mikrokontrolera ATmega328P

Nazwa pinu na płytce Arduino	Pin mikrokontrolera ATmega328P
RST	1
RX/D0	2
TX/D1	3
D2	4
D3	5
D4	6
(tylko 5 V)	7
GND	8
D5	11
D6	12
D7	13
D8	14
D9	15
D10	16
D11	17
D12	18
D13	19
(tylko 5 V)	20
AREF	21
GND	22
A0	23
A1	24
A2	25
A3	26
A4	27
A5	28

Aby uniknąć pomyłek, część sklepów (w tym adafruit i Freetronics) oferuje specjalne etykiety, które można przykleić do mikrokontrolera. Przykład takiej naklejki na mikrokontrolerze pokazano na rysunku 11.10 (etykieta jest dostępna na stronie *http://www.freetronics.com/mculabel/*).

Rysunek 11.10. Etykieta z oznaczeniami pinów

Uruchamianie szkicu testowego

Czas wysłać pierwszy szkic na platformę Arduino. Pierwszy szkic będzie bardzo prosty — spowoduje miganie diody LED:

```
// Projekt nr 37 — budowa własnej platformy Arduino na płytce uniwersalnej

void setup()
{
   pinMode(13, OUTPUT);
}

void loop()
{
   digitalWrite(13, HIGH);
   delay(1000);
   digitalWrite(13, LOW);
   delay(1000);
}
```

Szkic można wysłać na platformę Arduino na jeden z trzech sposobów.

Wymiana mikrokontrolera

Najtańszym sposobem wysłania szkicu jest usunięcie mikrokontrolera z oryginalnej płytki Arduino, umieszczenie na tej płytce mikrokontrolera z własnoręcznie budowanej platformy, wysłanie szkicu i ponowna wymiana mikrokontrolerów.

Aby bezpiecznie wyjąć mikrokontroler z płytki Arduino, należy użyć szczypiec do układów scalonych (patrz rysunek 11.11).

Rysunek 11.11. Do wyjęcia mikrokontrolera należy użyć szczypiec do układów scalonych

Podczas usuwania mikrokontrolera należy zadbać o *równomierne*, powolne wyciąganie obu końców układu — nie warto się spieszyć! Usuwanie komponentu może być dość trudne, ale ostatecznie mikrokontroler powinien dać się bezpiecznie oderwać od płytki.

Podczas umieszczania mikrokontrolera na płytce uniwersalnej lub na płytce Arduino może być konieczne nieznaczne dogięcie pinów, tak aby były ułożone prostopadle do mikrokontrolera i aby łatwiej można je było wsunąć w otwory na płytce. W tym celu należy umieścić jedną stronę komponentu na płaskiej powierzchni i delikatnie docisnąć układ (patrz rysunek 11.12). To samo należy zrobić z pinami po drugiej stronie układu.

Rysunek 11.12. Doginanie pinów mikrokontrolera

I wreszcie po ponownym umieszczeniu oryginalnego mikrokontrolera na płytce Arduino warto sprawdzić, czy końcówka oznaczona wcięciem znajduje się z odpowiedniej strony (patrz rysunek 11.13).

Rysunek 11.13. Prawidłowa orientacja mikrokontrolera na płytce Arduino

Połączenie z istniejącą płytką Arduino

Do wysyłania szkiców do kontrolera na własnoręcznie zbudowanej płytce Arduino (na płytce uniwersalnej) można używać interfejsu USB na oryginalnej płytce Arduino Uno. Ta metoda ogranicza zużycie gniazda na płytce Arduino i pozwala zaoszczędzić sporo środków, ponieważ nie wymaga zakupu dodatkowego przewodu USB do programowania.

Wysłanie szkicu na mikrokontroler za pośrednictwem interfejsu USB wymaga wykonania następującej procedury:

1. Należy odłączyć zasilanie (jeśli zostało podłączone) od platformy Arduino na płytce uniwersalnej.

2. Należy odłączyć przewód USB i usunąć mikrokontroler z płytki Arduino Uno.
3. Należy połączyć pin cyfrowy nr 0 na płytce Arduino z pinem nr 2 mikrokontrolera ATmega328P na płytce uniwersalnej oraz pin cyfrowy nr 1 na płytce Arduino z pinem nr 3 mikrokontrolera ATmega328P.
4. Należy połączyć piny 5 V i GND na płytce Arduino Uno z odpowiednimi otworami na płytce uniwersalnej.
5. Należy połączyć pin RST na płytce Arduino z pinem nr 1 mikrokontrolera ATmega328P.
6. Należy podłączyć przewód USB do płytki Arduino Uno.

Od tego momentu system powinien działać tak, jakby był zwykłą płytką Arduino Uno, zatem powinno być możliwe wysyłanie szkiców do mikrokontrolera na płytce uniwersalnej i (w razie konieczności) korzystanie z monitora portu szeregowego.

Stosowanie przewodu FTDI do programowania

Ostatnia metoda jest najprostsza, ale wymaga zakupu przewodu USB do programowania nazywanego *przewodem FTDI* (ponieważ układ odpowiedzialny za obsługę interfejsu USB na płytce Arduino został wyprodukowany przez firmę FTDI). Przed zakupem przewodu FTDI warto sprawdzić, czy oferowany produkt to model dla napięcia 5 V (przewód dla napięcia 3,3 V nie będzie działał prawidłowo). Opisywany przewód (patrz rysunek 11.14) jest z jednej strony zakończony wtyczką USB, a z drugiej strony gniazdem na sześć pinów. Końcówka z wtyczką USB zawiera obwód działający analogicznie do interfejsu USB na płytce Arduino Uno. Gniazdo na sześć pinów należy połączyć ze złączem pinowym pokazanym na rysunkach 11.7 i 11.8.

Rysunek 11.14. Przewód FTDI

Podczas łączenia tego przewodu należy sprawdzić, czy strona z czarnym przewodem jest podłączona do pinu GND złącza na płytce uniwersalnej. Po połączeniu przewód zapewnia zasilanie obwodu (tak jak przewód USB w przypadku oryginalnej płytki Arduino).

Przed wysłaniem szkicu lub otwarciem okna monitora portu szeregowego należy jeszcze zmienić typ płytki na Arduino Duemilanove lub Nano w/ ATmega328, wybierając opcję *Narzędzia/Płytka* i wskazując odpowiedni mikrokontroler (patrz rysunek 11.15).

Rysunek 11.15. Zmiana typu płytki w środowisku Arduino IDE

Po wybraniu metody wysyłania szkiców warto sprawdzić działanie tego rozwiązania, wysyłając szkic dla projektu nr 37. Potrafimy już projektować bardziej złożone obwody na bazie płytki uniwersalnej, dzięki czemu możemy budować dalsze projekty mniejszym kosztem. Jeśli dodatkowo opanujemy sztukę przygotowywania własnych płytek drukowanych, możemy nawet budować trwałe projekty.

Bogata rodzina płytek Arduino

Mimo że w tej książce jest używana wyłącznie płytka Arduino Uno, warto pamiętać o istnieniu wielu alternatywnych płytek zgodnych ze standardem Arduino. Poszczególne modele różnią się wymiarami, liczbą pinów wejściowych i wyjściowych, ilością pamięci dla szkiców i ceną.

Jedną z najważniejszych różnic dzielących poszczególne modele jest stosowany mikrokontroler. Modele aktualnie dostępne na rynku najczęściej są budowane na bazie mikrokontrolerów ATmega328P lub ATmega2560, natomiast w wersji Due stosuje się wersje o większej mocy obliczeniowej. Najważniejsze różnice pomiędzy stosowanymi układami (w tym pomiędzy dwiema wersjami mikrokontrolera ATmega328P) podsumowano w tabeli 11.2.

Tabela 11.2. Porównanie mikrokontrolerów

	ATmega328P	**ATmega328P SMD**	**ATmega2560**	**SAM3X8E**
Możliwość wymiany?	Tak	Nie	Nie	Nie
Częstotliwość taktowania	16 MHz	16 MHz	16 MHz	84 MHz
Napięcie zasilania	5 V	5 V	5 V	3,3 V
Liczba pinów cyfrowych	14 (6 obsługujących PWM)	14 (6 obsługujących PWM)	54 (14 obsługujących PWM)	54 (12 obsługujących PWM)
Liczba pinów analogowych	6	8	16	12
Natężenie prądu stałego na jeden pin wejścia-wyjścia	40 mA	40 mA	40 mA	Od 3 do 15 mA
Dostępna pamięć flash	31,5 kB	31,5 kB	248 kB	512 kB
Wielkość pamięci EEPROM	1 kB	1 kB	4 kB	Brak pamięci EEPROM
Wielkość pamięci SRAM	2 kB	2 kB	8 kB	96 kB

Do najważniejszych parametrów porównywanych przy okazji wyboru mikrokontrolerów zgodnych ze standardem Arduino należą rodzaje i objętość dostępnej pamięci. Poniżej wymieniono trzy rodzaje pamięci:

- *Pamięć flash* to przestrzeń dostępna dla szkicu skompilowanego i wysłanego za pośrednictwem środowiska IDE.
- *Pamięć EEPROM* (od ang. *electrically erasable programmable read-only memory*) to niewielka pamięć używana do przechowywania zmiennych (więcej informacji na ten temat można znaleźć w rozdziale 16.).
- *Pamięć SRAM* (od ang. *static random access memory*) to przestrzeń dostępna dla zmiennych używanych w programach.

UWAGA *Istnieje wiele płytek zgodnych ze standardem Arduino i alternatywnych względem modelu Uno — modele opisane w tym rozdziale to zaledwie wierzchołek góry lodowej. Jeśli planujesz realizację wielkich lub złożonych projektów, zachęcam Cię do zapoznania się z możliwościami większych płytek Mega. Z kolei jeśli będziesz potrzebował zaledwie kilku pinów wejścia-wyjścia i będziesz chciał zrealizować tylko jeden, trwały projekt, polecam płytkę Nano lub nawet jeszcze prostszy model LilyPad.*

W poniższych punktach opiszę kilka najbardziej popularnych płytek Arduino.

Płytka Arduino Uno

Model Uno jest obecnie postrzegany jako standardowa płytka Arduino. Wszystkie moduły dodatkowe dla platformy Arduino powinny być zgodne właśnie z tym modelem. Model Uno jest powszechnie uważany za najprostszego w użyciu przedstawiciela rodziny Arduino z uwagi na wbudowany interfejs USB i możliwość usunięcia mikrokontrolera.

Płytka Freetronics Eleven

Wiele płytek dostępnych na rynku emuluje działanie modelu Arduino Uno, a część z nich oferuje nawet usprawnienia względem standardowego projektu. Jedną z takich płytek jest Freetronics Eleven (patrz rysunek 11.16).

Rysunek 11.16. Płytka Freetronics Eleven

Mimo że płytka Eleven jest w pełni zgodna z modelem Arduino Uno, oferuje wiele usprawnień, które znacznie podnoszą wartość tego produktu. Pierwszym udoskonaleniem jest duży obszar płytki prototypowej dostępny bezpośrednio pod pinami wejścia-wyjścia. Dzięki temu można konstruować własne obwody bezpośrednio na głównej płytce, oszczędzając przestrzeń i pieniądze, ponieważ nie ma potrzeby kupowania dodatkowego modułu płytki prototypowej.

Po drugie, piny TX/RX, zasilania i diod LED umieszczono przy prawej krawędzi płytki, dzięki czemu dostęp do tych pinów jest możliwy nawet po zainstalowaniu dodatkowego modułu na płytce. I wreszcie na płytce zamontowano gniazdo micro-USB, które zajmuje dużo mniej miejsca niż standardowe gniazdo USB stosowane na płytce Uno. Dzięki temu projektowanie własnych modułów jest znacznie prostsze, ponieważ nie wymaga uwzględniania tak dużej przestrzeni zajmowanej przez gniazdo USB. Płytka jest dostępna na stronie *http://www.freetronics.com/products/eleven/*.

Płytka Freeduino

Płytka Freeduino jest wynikiem projektu open source, którego celem jest przygotowanie i publikacja plików ułatwiających konstruowanie własnych płytek zgodnych ze standardem Arduino. Jednym z najbardziej popularnych projektów tego typu jest zestaw zgodny z płytką Duemilanove (patrz rysunek 11.17).

Rysunek 11.17. Płytka Freeduino zgodna ze standardem Duemilanove

Płytka Freeduino umożliwia realizację wszystkich projektów opisanych w tej książce. Do największych zalet tego modelu należą niska cena i wyjątkowe uczucie satysfakcji po samodzielnym skonstruowaniu płytki. Zestaw do budowy płytki Freeduino jest dostępny na stronie *http://www.seeedstudio.com/*.

Płytka Boarduino

Płytka Boarduino jest minimalistyczną realizacją koncepcji Arduino stworzoną z myślą o stosowaniu na płytkach uniwersalnych. Płytka jest oferowana w formie zestawu wymagającego samodzielnego połączenia i lutowania. Gotową płytkę Boarduino pokazano na rysunku 11.18.

Rysunek 11.18. Gotowa do użycia płytka Boarduino

Zestaw Boarduino jest dostępny na stronie sklepu *http://www.adafruit.com/*. Produkt oferuje wszystkie funkcje zestawu Freeduino, ale osobne zasilanie jest niezbędne, gdy obwód wymaga prądu o natężeniu przekraczającym 500 mA.

Płytka Arduino Nano

Model Nano doskonale sprawdza się w sytuacji, gdy potrzebujemy małej, gotowej do pracy płytki zgodnej ze standardem Arduino. Płytka Nano jest co prawda mała, ale oferuje spore możliwości. Model zaprojektowano z myślą o montażu na płytce uniwersalnej (patrz rysunek 11.19).

Rysunek 11.19. Płytka Arduino Nano

Płytka Arduino Nano mierzy zaledwie 18×43 mm, a mimo to oferuje wszystkie funkcje znane z płytek Boarduino i Freeduino. Co więcej, na płytce zastosowano mikrokontroler ATmega328P w wersji SMD, która udostępnia dwa dodatkowe piny analogowe (A6 i A7). Model Nano jest dostępny na stronie *http://www.gravitech.us/arna30wiatp.html*.

Płytka Arduino LilyPad

Płytkę LilyPad zaprojektowano z myślą o integracji w ramach innych projektów, na przykład urządzeń elektronicznych wszywanych w ubrania. Okazuje się, że płytkę LilyPad można myć wodą i delikatnym detergentem, zatem doskonale nadaje się na przykład do przygotowania podświetlanej bluzy sportowej. Projekt tego modelu jest niepowtarzalny (patrz rysunek 11.20).

Łączenie z pinami wejścia-wyjścia na płytce LilyPad wymaga lutowania przewodów do tej płytki, zatem opisywany model lepiej sprawdza się w przypadku trwałych projektów. Z racji minimalistycznego projektu układ nie zawiera elementów regulujących napięcie, zatem to użytkownik odpowiada za zapewnienie zasilania prądem o napięciu od 2,7 do 5,5 V. Płytka LilyPad nie zawiera też interfejsu USB, zatem wysyłanie szkiców wymaga użycia przewodu FTDI 5 V. Model Arduino LilyPad jest dostępny niemal we wszystkich sklepach oferujących płytki Arduino.

Rysunek 11.20. Płytka Arduino LilyPad

Płytka Arduino Mega 2560

W razie wykorzystania wszystkich pinów wejścia-wyjścia na płytce Arduino Uno lub braku miejsca na większe szkice warto rozważyć zakup modelu Mega 2560. Płytka Mega 2560 jest znacznie większa od standardowej płytki — mierzy 109 na 53 mm (patrz rysunek 11.21).

Rysunek 11.21. Płytka Arduino Mega 2560

Płytka Mega 2560, mimo że jest dużo większa od modelu Uno, oferuje możliwość montażu większości modułów dodatkowych dla platformy Arduino. Płytka Mega dodatkowo umożliwia stosowanie dużych płytek prototypowych dla większych projektów, które trudno byłoby zmieścić na płytkach dla modelu Uno. Ponieważ na płytce Mega zastosowano mikrokontroler ATmega2560, dostępna ilość pamięci i liczba pinów wejścia-wyjścia (patrz tabela 11.2) jest dużo większa niż w przypadku standardowej płytki Uno. Co więcej, cztery odrębne linie komunikacji szeregowej zwiększają możliwości transmisji danych. Model Mega 2560 jest dostępny niemal we wszystkich sklepach oferujących płytki Arduino.

Płytka Freetronics EtherMega

Model EtherMega (patrz rysunek 11.22) jest doskonałym wyborem dla konstruktorów, którzy potrzebują płytki Arduino Mega 2560, modułu kart pamięci microSD i modułu sieciowego umożliwiającego łączenie urządzeń z internetem. EtherMega oferuje wszystkie te komponenty na jednej płytce, która przy tym jest tańsza niż suma kosztów poszczególnych produktów. Płytka EtherMega jest dostępna na stronie *http://www.freetronics.com/ethermega/*.

Rysunek 11.22. Płytka Freetronics EtherMega

Płytka Arduino Due

Procesor 84 MHz umożliwia dużo szybsze wykonywanie szkiców niż w przypadku najszybszych płytek Arduino dostępnych na rynku. Jak widać na rysunku 11.23, płytka Arduino Due pod wieloma względami przypomina model Arduino Mega 2560, ale oferuje dodatkowy port USB dla urządzeń zewnętrznych i cechuje się nieco innym układem pinów.

Rysunek 11.23. Płytka Arduino Due

Co więcej, płytka Due oferuje ponad 16 razy więcej przestrzeni pamięciowej niż model Uno, zatem umożliwia tworzenie naprawdę złożonych i szczegółowych szkiców. Warto jednak pamiętać, że płytka Due jest zasilana prądem o napięciu 3,3 V, zatem urządzenia podłączone do pinów cyfrowych i analogowych nie mogą dostarczać napięcia przekraczającego ten poziom. Mimo tych ograniczeń korzyści wynikające ze stosowania płytki Due znacznie przewyższają koszty zmiany pozostałych urządzeń.

UWAGA *Podczas zakupu drugiej płytki Arduino lub dodatkowych akcesoriów warto wybierać sprawdzonych dostawców, którzy oferują odpowiednie wsparcie i gwarancję. W internecie aż roi się od tańszych zamienników, których wyjątkowo niska cena wynika właśnie z unikania odpowiedzialności za uszkodzone lub niezgodne ze specyfikacją produkty.*

SPRZĘT OPEN SOURCE

Projekt rozwiązań sprzętowych stosowanych na płytkach Arduino jest dostępny publicznie, zatem każdy może budować, modyfikować, sprzedawać i stosować własne wersje tej platformy. Ta forma dystrybucji jest określana mianem sprzętu open source — to stosunkowo nowa koncepcja, która jest zupełnym przeciwieństwem ochrony praw autorskich i prawnego zabezpieczania własności intelektualnej. Zespół odpowiedzialny za rozwój platformy Arduino zdecydował, że projekt tego systemu będzie dostępny za darmo, dzięki czemu trafi do szerokiej społeczności konstruktorów.

Także wiele organizacji produkujących akcesoria dla platformy Arduino i modyfikacje oryginalnych płytek postępuje zgodnie z ideą sprzętu open source i na tych samych zasadach udostępnia swoje projekty. Takie rozwiązanie znacznie przyspiesza proces doskonalenia produktów w porównaniu z tradycyjnym modelem, w którym jedna organizacja pracuje nad rozwojem swojej platformy.

Co dalej?

Ten rozdział powinien poszerzyć Twoją wiedzę na temat rodzajów dostępnego sprzętu, w tym systemu Arduino samodzielnie budowanego na bazie płytki uniwersalnej. Omówiłem elementy składające się na projekt systemu Arduino oraz sposób budowy własnego obwodu zgodnego z tym standardem. Dowiodłem, że można budować wiele prototypów na bazie tej platformy bez konieczności każdorazowego kupowania płytek Arduino. Opisałem też różne płytki Arduino dostępne na rynku, zatem każdy może teraz wybrać płytkę, która najlepiej spełnia jego oczekiwania. I wreszcie wyjaśniłem ideę sprzętu open source, która przyświeca twórcom platformy Arduino.

W następnym rozdziale wprowadzę techniki sterowania różnymi silnikami i omówię pierwsze kroki projektu polegającego na budowie własnego robota gąsienicowego sterowanego przez system Arduino!

12

Silniki i ruch

W tym rozdziale:

- użyjesz silnika wykonawczego (serwomotoru) do budowy termometru analogowego;
- dowiesz się, jak sterować szybkością i kierunkiem działania silników elektrycznych;
- użyjesz modułu silnika dla platformy Arduino;
- rozpoczniesz budowę ruchomego robota z napędem gąsienicowym;
- zastosujesz proste mikroprzełączniki w celu unikania kolizji;
- użyjesz podczerwonych i ultradźwiękowych czujników odległości do unikania kolizji.

Wprawianie urządzeń w ruch za pomocą silników wykonawczych

Silnik wykonawczy (nazywany też serwomotorem; ang. *servo*, *servomechanism*) zawiera silnik elektryczny, któremu można wydać polecenie obrotu do określonej pozycji (wyrażonej jako kąt obrotu). Silnik wykonawczy może służyć na przykład do sterowania zdalnie sterowanym samochodem — wystarczy połączyć ten silnik z niewielkim ramieniem lub dźwignią. Przykładem ramienia jest wskazówka zegara analogowego. Silnik wykonawczy i trzy rodzaje ramion pokazano na rysunku 12.1.

Rysunek 12.1. Silnik wykonawczy i różne ramiona

Wybór silnika wykonawczego

Przy wyborze silnika wykonawczego należy mieć na uwadze następujące parametry:

- **Szybkość**, czyli czas potrzebny na obrót (wyrażany najczęściej w liczbie sekund potrzebnych do obrotu o określony kąt).
- **Zakres obrotu**, czyli maksymalny kąt obrotu silnika — na przykład 180 stopni (połowa pełnego obrotu) lub 360 stopni (jeden pełny obrót).
- **Natężenie**, czyli natężenie prądu elektrycznego potrzebne do działania silnika wykonawczego. W przypadku sterowania silnikiem wykonawczym za pomocą systemu Arduino może być konieczne użycie dodatkowego, zewnętrznego zasilacza.

- **Moment obrotowy**, czyli siła generowana przez silnik wykonawczy podczas obrotu. Im większy jest moment obrotowy, tym większa może być masa elementów poruszanych przez silnik. Moment obrotowy zwykle rośnie proporcjonalnie do natężenia prądu zasilającego silnik wykonawczy.

Na rysunku 12.1 pokazano silnik wykonawczy hexTronik HXT900. Ten niedrogi komponent oferuje możliwość obrotu o 180 stopni (patrz rysunek 12.2).

Rysunek 12.2. Przykładowy przedział obrotu silnika wykonawczego

Łączenie silnika wykonawczego

Łączenie silnika wykonawczego z płytką Arduino jest proste, ponieważ wymaga użycia tylko trzech przewodów. W przypadku stosowania silnika HXT900 najciemniejszy przewód należy połączyć z pinem GND, środkowy przewód z pinem 5 V, a najjaśniejszy przewód (*impulsowy*) z pinem cyfrowym. Podczas łączenia innego silnika wykonawczego warto sprawdzić sposób łączenia w karcie produktu.

Uruchamianie silnika wykonawczego

Spróbujmy teraz uruchomić silnik wykonawczy. W tym szkicu silnik wykonawczy będzie się obracał w całym dostępnym zakresie. Silnik należy podłączyć do płytki Arduino zgodnie z wcześniejszym opisem (przewód impulsowy należy połączyć z pinem cyfrowym nr 4), a następnie wpisać i wysłać na płytkę Arduino szkic z listingu 12.1.

Listing 12.1. Szkic demonstrujący działanie silnika wykonawczego

```
// Listing 12.1

#include <Servo.h>
Servo myservo;

void setup()
{
   myservo.attach(4);
}
```

```
void loop()
{
   myservo.write(180);
   delay(1000);
   myservo.write(90);
   delay(1000);
   myservo.write(0);
   delay(1000);
}
```

W powyższym szkicu użyto biblioteki *servo* dołączonej w środowisku Arduino IDE w celu utworzenia obiektu silnika wykonawczego:

```
#include <Servo.h>
Servo myservo;
```

W funkcji `void setup()` wskazano systemowi Arduino, który pin cyfrowy będzie używany do sterowania silnikiem wykonawczym:

```
myservo.attach(4); // do sterowania silnikiem będzie służył pin cyfrowy nr 4
```

Wymuszenie ruchu silnika wykonawczego wymaga już tylko następującego wyrażenia:

```
myservo.write(x);
```

Parametr `x` reprezentuje liczbę całkowitą z przedziału od 0 do 180 — czyli docelowe położenie silnika wykonawczego. Po uruchomieniu szkicu z listingu 12.1 silnik będzie się obracał do skrajnych pozycji. Obrót silnika będzie wstrzymywany na jedną sekundę zarówno przy minimalnym i maksymalnym kącie obrotu (odpowiednio 0 i 180 stopni), jak i w punkcie środkowym (90 stopni). Podczas obserwacji ruchów silnika wykonawczego łatwo zauważyć, że pozycja reprezentowana przez 180 stopni znajduje się po lewej stronie, natomiast pozycja reprezentowana przez 0 stopni znajduje się po prawej stronie.

Oprócz pchania i ciągnięcia fizycznych obiektów silniki wykonawcze mogą służyć także do komunikowania danych w sposób znany z analogowych wskaźników. Silnik wykonawczy może służyć na przykład do wyświetlania wskazań termometru analogowego (patrz projekt nr 38).

Projekt nr 38: budowa termometru analogowego

Za pomocą silnika wykonawczego i czujnika temperatury TMP36 (stosowanego we wcześniejszych rozdziałach) można zbudować termometr analogowy. Urządzenie będzie mierzyło temperaturę i konwertowało ten pomiar na kąt z przedziału od 0 do 180 stopni odpowiadający temperaturze z przedziału odpowiednio od 0 do 30 stopni Celsjusza. Silnik wykonawczy będzie ustawiał wskazówkę zgodnie z aktualną temperaturą.

Sprzęt

Projekt wymaga bardzo niewielu komponentów:

- czujnika temperatury TMP36;
- płytki uniwersalnej;
- małego silnika wykonawczego;
- różnych przewodów;
- płytki Arduino i przewodu USB.

Schemat

Także schemat obwodu dla tego projektu jest bardzo prosty (patrz rysunek 12.3).

Rysunek 12.3. Schemat obwodu dla projektu nr 38

Szkic

Szkic określa aktualną temperaturę otoczenia przy użyciu tej samej metody co w projekcie nr 8. Odczytana temperatura jest następnie konwertowana na kąt obrotu silnika wykonawczego.

W środowisku Arduino IDE należy wpisać i wysłać na płytkę następujący szkic:

```
// Projekt nr 38 — budowa termometru analogowego
float  voltage = 0;
float   sensor = 0;
float currentC = 0;
int      angle = 0;

#include <Servo.h>
Servo myservo;

void setup()
{
   myservo.attach(4);
}

int calculateservo(float temperature)
{
   float resulta;
   int resultb;
   resulta = -6 * temperature;
   resulta = resulta + 180;
   resultb = int(resulta);
   return resultb;
}

void loop()
{
   // odczytuje bieżącą temperaturę
   sensor = analogRead(0);
   voltage = (sensor*5000)/1024;
   voltage = voltage-500;
   currentC = voltage/10;

   // wyświetla bieżącą temperaturę za pomocą silnika wykonawczego
   angle = calculateservo(currentC);
   // konwertuje temperaturę na położenie silnika wykonawczego
   if (angle>=0 && angle <=30)
   {
      myservo.write(angle); // ustawia odpowiedni kąt obrotu silnika wykonawczego
      delay(1000);
   }
}
```

Większość elementów tego szkicu była stosowana już wcześniej; jedyną nowością jest funkcja `calculateservo()`. Funkcja konwertuje temperaturę na odpowiedni kąt obrotu silnika wykonawczego na podstawie następującego wzoru:

kąt = (–6 × temperatura) + 180

Aby efekt działania tego urządzenia był bardziej realistyczny, warto zastosować niewielką *tarczę ze skalą* i wskazówkę. Przykład takiej tarczy ze skalą pokazano na rysunku 12.4.

Rysunek 12.4. Skala i wskazówka pokazujące temperaturę wskazywaną przez termometr

Stosowanie silników elektrycznych

Następnym krokiem w procesie poznawania metod sterowania ruchem jest wprowadzenie małych silników elektrycznych. Niewielkie silniki znajdują wiele zastosowań, od małych wentylatorów po zabawki i kolejki elektryczne. Tak jak w przypadku silników wykonawczych, także tradycyjne silniki elektryczne mają kilka istotnych parametrów:

- **Napięcie zasilania**, które zwykle mieści się w przedziale od 3 do ponad 12 V.
- **Prąd bez obciążenia**, czyli natężenie prądu elektrycznego niezbędne (przy napięciu zasilania) do swobodnego obrotu wału, który nie jest połączony z żadnymi innymi obiektami.
- **Prąd zatrzymania silnika**, czyli natężenie, przy którym silnik próbuje obracać wałem, ale nie może tego zrobić z powodu zbyt dużego obciążenia.
- **Szybkość przy napięciu zasilacza**, czyli liczba obrotów wału na minutę (ang. *revolutions per minute — RPM*).

W najbliższym przykładzie zostanie użyty mały, niedrogi silnik elektryczny pracujący z szybkością 8540 obrotów na minutę przy napięciu zasilania 3 V (przykład silnika o takich parametrach pokazano na rysunku 12.5).

Rysunek 12.5. Mały silnik elektryczny

Do sterowania tym silnikiem zostanie użyty tranzystor, czyli komponent wprowadzony w rozdziale 3. Ponieważ silnik wymaga prądu elektrycznego o natężeniu 0,7 A (czyli więcej, niż może przekazywać tranzystor BC548), tym razem zostanie użyty tzw. tranzystor Darlingtona.

Tranzystor Darlingtona TIP120

Tranzystor Darlingtona może obsługiwać duże natężenia i napięcia prądu elektrycznego. Tranzystor Darlingtona TIP120 może przekazywać prąd o natężeniu nieprzekraczającym 5 A i maksymalnym napięciu 60 V, czyli o parametrach dużo wyższych od tych potrzebnych do sterowania niewielkim silnikiem elektrycznym. Symbol tranzystora TIP120 bardzo przypomina symbol tranzystora BC548 (z rysunku 12.6), jednak tranzystor TIP120 jest wyraźnie większy od modelu BC548.

Rysunek 12.6. Symbol tranzystora TIP120 stosowany na schematach obwodów

W tranzystorze TIP120 zastosowano model łączenia zgodny ze standardem TO-220 (patrz rysunek 12.7).

Jeśli patrzymy na tranzystor TIP120 od strony z nadrukowaną etykietą, pierwszy pin od lewej strony to baza (B), drugi (środkowy) pin to kolektor (C), a trzeci, skrajnie prawy pin to emiter (E). Także blaszka odpowiedzialna za odprowadzanie ciepła jest połączona z kolektorem.

Rysunek 12.7. Tranzystor TIP120

Projekt nr 39: sterowanie silnikiem

W tym projekcie sterowanie silnikiem będzie polegało na zmianie szybkości jego pracy.

Sprzęt

Projekt będzie wymagał następującego sprzętu:

- małego silnika elektrycznego zasilanego napięciem 3 V;
- rezystora 1 kΩ (R1);
- płytki uniwersalnej;
- diody 1N4004;
- tranzystora Darlingtona TIP120;

- dodatkowego źródła zasilania prądem 3 V;
- różnych przewodów;
- płytki Arduino i przewodu USB.

Podczas budowy projektów wymagających użycia silników elektrycznych należy stosować dodatkowe źródła zasilania, ponieważ płytka Arduino nie może dostarczyć prądu o natężeniu niezbędnym do pracy silnika. Silnik, który zatrzymuje się wskutek zbyt dużego obciążenia, pobiera natężenie osiągające tzw. *prąd zatrzymania* — w niektórych przypadkach może ono przekraczać 1 amper. To zdecydowanie więcej, niż może dostarczyć płytka Arduino — próba zasilania silnika przez Arduino może doprowadzić do trwałego uszkodzenia płytki.

Rysunek 12.8. Opakowanie na dwie baterie AA

Jednym z najprostszych rozwiązań jest zewnętrzny zestaw baterii. Do zapewnienia zasilania prądem 3 V wystarczy opakowanie na dwie baterie AA z wyprowadzonymi przewodami (patrz rysunek 12.8).

Schemat

Obwód należy połączyć zgodnie ze schematem pokazanym na rysunku 12.9.

Rysunek 12.9. Schemat obwodu dla projektu nr 39

Szkic

W tym projekcie szybkość działania silnika będzie stopniowo zwiększana od zatrzymania do wartości maksymalnej i ponownie do zatrzymania. W środowisku IDE należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 39 — sterowanie silnikiem

void setup()
{
   pinMode(5, OUTPUT);
}

void loop()
{
❶     for (int a=0; a<256; a++)
      {
         analogWrite(5, a);
❷        delay(100);
      }
❸        delay(5000);
❹        for (int a=255; a>=0; a--)
      {
         analogWrite(5,a);
         delay(100);
      }
      delay(5000);
}
```

Szybkością działania silnika można sterować, stosując metodę modulacji szerokości impulsu (wyjaśnioną przy okazji omawiania projektu nr 3). Warto pamiętać, że technika ta jest dostępna tylko w przypadku pinów cyfrowych nr 3, 5, 6, 9, 10 i 11. Metoda polega na dostarczaniu do silnika zasilania tylko przez pewien czas w określonym okresie — im dłuższy jest ten czas, tym szybciej pracuje silnik, ponieważ czas zasilania jest dłuższy niż czas, w którym silnik jest wyłączony. W wierszu ❶ szybkość działania silnika jest stopniowo zwiększana (początkowo silnik jest zatrzymany). Przyspieszenie można kontrolować, zmieniając opóźnienie w wierszu ❷. W wierszu ❸ silnik działa z maksymalną szybkością i utrzymuje tę szybkość przez 5 sekund. Następnie, począwszy od wiersza ❹, proces jest odwracany i silnik stopniowo zwalnia.

UWAGA *Podczas uruchamiania silnik może piszczeć (wydawać dźwięk podobny do pociągu elektrycznego lub tramwaju ruszającego ze stacji lub przystanku). To zjawisko jest zupełnie normalne i nie powinno nas niepokoić.*

W opisanym obwodzie dioda jest używana w tym samym celu co przekaźnik pokazany na rysunku 3.19, czyli do ochrony obwodu. W momencie wyłączenia zasilania silnika w obwodzie (a konkretnie na cewce silnika) przez krótki czas

występuje tzw. prąd błądzący, który należy gdzieś skierować. Dioda powoduje, że prąd błądzący wielokrotnie przechodzi przez cewkę aż do ostatecznego zaniknięcia (zamiany na niewielką ilość energii cieplnej).

Projekt nr 40: budowa robota gąsienicowego i sterowanie tym robotem

Mimo że sterowanie szybkością pracy silnika elektrycznego jest ciekawym i przydatnym rozwiązaniem, jeszcze ciekawsze jest jednoczesne sterowanie dwoma silnikami (zarówno szybkością, jak i kierunkiem obrotu wałów). Celem tego projektu jest rozpoczęcie prac nad skonstruowaniem robota gąsienicowego (przypominającego czołg); zadanie będzie stopniowo realizowane także w kolejnych rozdziałach. W ramach tego projektu przygotujemy najważniejsze elementy robota i podstawowy układ sterowania.

Robot będzie używał dwóch silników, po jednym na gąsienicę, dzięki czemu będzie mógł bezpiecznie pokonywać niewielkie przeszkody i obracać się w miejscu. Użytkownik będzie sterował szybkością i kierunkiem ruchu robota; w dalszej części tego rozdziału i w kolejnych rozdziałach dodatkowo wyposażymy robota w elementy umożliwiające unikanie kolizji i mechanizmy zdalnego sterowania. Po zakończeniu wszystkich projektów opisanych w tej książce będziesz dysponował solidną wiedzą potrzebną do realizacji własnych pomysłów i budowy alternatywnych wariantów tego zaawansowanego urządzenia.

Sprzęt

Realizacja tego projektu będzie wymagała następujących urządzeń:

- zestawu do konstruowania podwozia Pololu RP5;
- płyty montażowej dla podwozia Pololu RP5;
- sześciu alkalicznych baterii AA;
- baterii 9 V do zasilania gniazda DC;
- modułu silników DFRobot 2A;
- płytki Arduino i przewodu USB.

Podwozie

Podstawą każdego robota jest solidne podwozie utrzymujące silniki, układ przeniesienia napędu i źródło zasilania. Robot sterowany przez system Arduino dodatkowo musi oferować miejsce dla płytki Arduino i różnych elementów zewnętrznych.

Na rynku są dostępne różne modele podwozi; w tym projekcie zostanie użyte podwozie z gąsienicami napędzanymi dwoma silnikami — Pololu RP5 (patrz rysunek 12.10).

Rysunek 12.10. Podwozie robota gąsienicowego

Dwa źródła zasilania

Podwozie Pololu oferuje miejsce na sześć baterii AA, których będziemy używali w roli źródła zasilania dla silników (patrz rysunek 12.11). Komora baterii znajduje się na spodzie podwozia (pomiędzy silnikami), co zapewnia niski punkt ciężkości robota.

Rysunek 12.11. Komora na sześć baterii AA

Mimo że komora na baterie pokazana na rysunku 12.11 jest dość duża, będzie potrzebne jeszcze jedno źródło zasilania dla samej płytki Arduino, tak aby system robota mógł działać nawet w razie wyczerpania baterii zasilających silniki. W tym projekcie płytka Arduino będzie zasilana baterią 9 V, którą można połączyć z gniazdem zasilania na tej płytce za pomocą takiego przewodu jak przedstawiony na rysunku 12.12.

Rysunek 12.12. Przewód używany do łączenia baterii z płytką Arduino

Płyta montażowa

Ostatnim elementem podwozia jest *płyta montażowa* (patrz rysunek 12.13).

Rysunek 12.13. Płyta montażowa

Płyta montażowa, którą należy zainstalować na górze podwozia, umożliwia montaż elementów przy użyciu dystansów i *śrubek* M3. (Śrubki, dystansy i niezbędne podkładki można bez trudu znaleźć w sklepach z częściami dla robotyków oraz w dużych sklepach narzędziowych). Na rysunku 12.14 pokazano płytę montażową z przykręconymi dystansami i zainstalowaną płytką Arduino.

Rysunek 12.14. Płytka Arduino zainstalowana na płycie montażowej

Schemat

Ostatnim krokiem procesu przygotowywania sprzętu dla tego projektu jest skonstruowanie obwodu potrzebnego do sterowania oboma silnikami zamontowanymi na podwoziu. Mimo że dla każdego z silników można by zastosować obwód pokazany na rysunku 12.9, takie rozwiązanie uniemożliwiłoby nam sterowanie kierunkiem pracy silników, a samo łączenie dwóch silników w ten sposób byłoby dość niewygodne. Tym razem zostanie użyty *moduł silników*. Moduł silników zawiera obwody niezbędne do obsługi większego natężenia prądu elektrycznego pobieranego przez silniki i realizuje przekazywane przez system Arduino polecenia dotyczące szybkości i kierunku pracy silników. Podczas budowy naszego robota zostanie użyty moduł 2A Motor Shield firmy DFRobot (*http://www.dfrobot.com/*) dla platformy Arduino (patrz rysunek 12.15).

Rysunek 12.15. Moduł silników DFRobot

Łączenie modułu silników

Łączenie modułu silników jest dość proste — należy najpierw podłączyć przewody wyprowadzone z komory baterii do bloku zacisków w lewym dolnym rogu modułu (patrz rysunek 12.16). Czarny przewód (ujemny) należy umieścić w otworze po prawej stronie; czerwony przewód w otworze po lewej stronie.

Należy teraz podłączyć pary przewodów dla obu silników. Warto się upewnić, że przewody odpowiednich kolorów zostały umieszczone we właściwych otworach bloku zacisków (patrz rysunek 12.17).

Rysunek 12.16. Przewody zasilające podłączone do zacisków na płytce modułu silnika

Rysunek 12.17. Łączenie silników

Łączenie zworek

Ostatnim krokiem procedury przygotowywania modułu silników jest odpowiednie ustawienie zworek. Warto przyjrzeć się nieco bliżej przestrzeni pomiędzy złączami zasilania a dolnym wierszem gniazd na płytce tego modułu — widać tam sześć pinów i dwie czarne zworki. Zworki należy umieścić poziomo, tak aby obejmowały cztery piny po lewej stronie (patrz rysunek 12.18). I wreszcie należy sprawdzić, czy cztery pary pinów PWM są połączone zworkami w orientacji pionowej (patrz rysunek 12.19).

Rysunek 12.18. Prawidłowe ustawienie zworek w sekcji zasilania

Rysunek 12.19. Prawidłowe ustawienie zworek trybów pracy

Jeśli przewody wychodzące z silników nie zostały oznaczone kolorami, po pierwszym uruchomieniu i sprawdzeniu, która orientacja powoduje ruch w jedną stronę, może być konieczna zmiana sposobu połączenia przewodów.

Po połączeniu przewodów i zworek, instalacji komory baterii, przymocowaniu płytki Arduino i modułu silników na płycie montażowej oraz skręceniu podwozia robot powinien wyglądać tak jak na rysunku 12.20.

Rysunek 12.20. Robot gąsienicowy gotowy do akcji!

Szkic

Czas spowodować, że robot będzie się poruszał. Warto zacząć od przygotowania funkcji, które uproszczą sterowanie robotem. Ponieważ robot będzie napędzany dwoma silnikami, szkic musi obsługiwać cztery możliwe ruchy:

- ruch w przód;
- ruch w tył;
- obrót w kierunku zgodnym z ruchem wskazówek zegara;
- obrót w kierunku przeciwnym do ruchu wskazówek zegara.

Moduł silników steruje każdym silnikiem za pośrednictwem dwóch pinów cyfrowych — jeden służy do sterowania szybkością przy użyciu metody modulacji szerokości impulsu (tak jak w projekcie nr 39), drugi określa kierunek obrotów wału silnika.

Szkic zawiera cztery funkcje obsługujące wymienione powyżej ruchy robota: `goForward()`, `goBackward()`, `rotateLeft()` i `rotateRight()`. Każda z tych funkcji otrzymuje na wejściu dwie wartości: czas ruchu wyrażony w milisekundach oraz szybkość ruchu reprezentowaną przez wartość z przedziału od 0 do 255 (i decydującą o szerokości impulsu PWM). Oznacza to, że aby robot poruszał się w przód przez 2 sekundy z maksymalną szybkością, należy użyć funkcji `goForward(2000,255)`.

W środowisku Arduino IDE trzeba wprowadzić i zapisać w pliku następujący szkic (na tym etapie nie należy jednak wysyłać tego szkicu na platformę Arduino):

```
// Projekt nr 40 — budowa robota gąsienicowego i sterowanie tym robotem

int m1speed=6; // piny cyfrowe używane do sterowania szybkością
int m2speed=5;
int m1direction=7; // piny cyfrowe używane do sterowania kierunkiem pracy silników
int m2direction=4;

void setup()
{
   pinMode(m1direction, OUTPUT);
   pinMode(m2direction, OUTPUT);
   delay(5000);
}

void goForward(int duration, int pwm)
{
❶  digitalWrite(m1direction,HIGH); // ruch w przód
   digitalWrite(m2direction,HIGH); // ruch w przód
   analogWrite(m1speed, pwm); // szybkość
   analogWrite(m2speed, pwm);
   delay(duration);
   analogWrite(m1speed, 0); // szybkość
   analogWrite(m2speed, 0);
}
```

```
void goBackward(int duration, int pwm)
{
❷   digitalWrite(m1direction,LOW); // ruch w tył
    digitalWrite(m2direction,LOW); // ruch w tył
    analogWrite(m1speed, pwm); // szybkość
    analogWrite(m2speed, pwm);
    delay(duration);
    analogWrite(m1speed, 0); // szybkość
    analogWrite(m2speed, 0);
}

void rotateRight(int duration, int pwm)
{
❸   digitalWrite(m1direction,HIGH); // ruch w przód
    digitalWrite(m2direction,LOW); // ruch w tył
    analogWrite(m1speed, pwm); // szybkość
    analogWrite(m2speed, pwm);
    delay(duration);
    analogWrite(m1speed, 0); // szybkość
    analogWrite(m2speed, 0);
}

void rotateLeft(int duration, int pwm)
{
❹   digitalWrite(m1direction,LOW); // ruch w tył
    digitalWrite(m2direction,HIGH); // ruch w przód
    analogWrite(m1speed, pwm); // szybkość
    analogWrite(m2speed, pwm);
    delay(duration);
    analogWrite(m1speed, 0); // szybkość
    analogWrite(m2speed, 0);
}

void loop()
{
    goForward(1000, 255);
    rotateLeft(1000, 255);
    goForward(1000, 255);
    rotateRight(1000, 255);
    goForward(1000, 255);
    goBackward(2000, 255);
    delay(2000);
}
```

W tym szkicu ustawiono kierunek obrotów wału każdego silnika za pomocą funkcji:

```
digitalWrite(m1direction,direction);
```

Parametr `direction` powinien mieć wartość `HIGH` dla ruchu w przód i `LOW` dla ruchu w tył. Oznacza to, że aby robot gąsienicowy poruszał się w przód, należy ustawić parametry ruchu obu silników w ten sam sposób (patrz wiersze ❶ i ❷). Do ustawiania szybkości działania silnika użyto następującego wywołania funkcji:

```
analogWrite(m1speed, pwm);
```

Wartość parametru `pwm` reprezentuje szybkość i musi się mieścić w przedziale od 0 do 255. Aby robot obracał się w miejscu, silniki powinny działać w przeciwnych kierunkach (patrz wiersze ❸ i ❹).

OSTRZEŻENIE *Przed wysłaniem szkicu na płytkę Arduino należy albo umieścić robota w bezpiecznym miejscu, albo ustawić w taki sposób, aby gąsienice nie miały kontaktu z podłożem; w przeciwnym razie robot w ciągu kilku sekund „ożyje" i zjedzie z biurka!*

Należy teraz wysłać szkic, usunąć przewód USB i podłączyć przewód komory baterii do gniazda zasilania płytki Arduino. Robot można następnie postawić na dywanie lub innej czystej powierzchni, aby sprawdzić, jak się porusza. Warto też samodzielnie poeksperymentować z funkcjami ruchu zaimplementowanymi w tym projekcie, aby lepiej zrozumieć wpływ opóźnień i ograniczeń czasowych na odległości pokonywane przez pojazd.

Wykrywanie kolizji

Skoro nasz robot potrafi się już poruszać, możemy przystąpić do implementacji prostych rozwiązań poprawiających inteligencję działania tego urządzenia, w tym czujników kolizji, które będą wykrywały zderzenia robota z innymi obiektami lub mierzyły odległość dzielącą robota od obiektów na jego drodze (aby unikać kontaktu). Do unikania kolizji zastosujemy trzy metody: mikroprzełączniki, czujniki podczerwieni i czujniki ultradźwiękowe.

Projekt nr 41: wykrywanie kolizji robota za pomocą mikroprzełącznika

Mikroprzełącznik (ang. *microswitch*) może działać jak prosty włącznik (podobny do tego stosowanego w rozdziale 4.), tyle że jest fizycznie większy od przycisku i zawiera dużą metalową dźwignię działającą jak siłownik (patrz rysunek 12.21).

Podczas instalacji mikroprzełącznika jeden przewód należy połączyć z dolnym stykiem, drugi przewód należy połączyć ze stykiem oznaczonym etykietą NO (standardowo otwarty), aby zagwarantować, że prąd będzie przepływał tylko

po naciśnięciu dźwigni. Mikroprzełącznik zostanie zamontowany z przodu naszego robota, zatem w momencie uderzenia w jakiś obiekt dźwignia zostanie naciśnięta, co spowoduje włączenie przepływu prądu i wymusi ruch robota w przeciwnym kierunku.

Rysunek 12.21. Przykład mikroprzełącznika

Schemat

Metoda łączenia mikroprzełącznika nie różni się od sposobu łączenia przycisku (patrz rysunek 12.22).

Rysunek 12.22. Schemat wykrywacza kolizji na bazie mikroprzełącznika

Szkic

Mikroprzełącznik został podłączony do portu przerwań (pinu cyfrowego nr 2). Na pierwszy rzut oka wydaje się, że najlepszym rozwiązaniem byłoby utworzenie funkcji wywoływanej przez to przerwanie, tak aby przez pewien czas robot poruszał się do tyłu, jednak opracowanie tej funkcji byłoby niemożliwe z racji

braku obsługi funkcji `delay()` w ciele funkcji wywoływanych przez przerwania. W tym przypadku należy zastosować zupełnie inny model.

Zamiast specjalnej funkcji wywoływanej przez przerwanie to istniejąca `goForward()` będzie zmieniać kierunek jazdy w razie spełnienia dwóch warunków dotyczących zmiennych logicznych `crash` i `move`. Jeśli zmienna `crash` ma wartość `true`, funkcja wymusza działanie silników w przeciwnym kierunku przez dwie sekundy i z mniejszą szybkością, aby robot mógł się wycofać z miejsca, w którym nastąpiła kolizja.

Ponieważ nie można stosować funkcji `delay()` w kodzie wywoływanym przez przerwanie, czas pracy silników (i wycofywania robota) będzie mierzony za pomocą funkcji `millis()` (wynik tej funkcji zostanie zapisany na początku ruchu i wielokrotnie w pętli aż do osiągnięcia różnicy). Jeśli różnica pomiędzy wartością początkową a bieżącą jest większa lub równa liczbie właściwej wybranemu okresowi, zmiennej `move` jest przypisywana wartość `false` i następuje zatrzymanie silników.

W środowisku Arduino IDE należy teraz wpisać i wysłać następujący szkic:

```
// Projekt nr 41 — wykrywanie kolizji robota za pomocą mikroprzełącznika

int m1speed=6; // piny cyfrowe używane do sterowania szybkością
int m2speed=5;
int m1direction=7; // piny cyfrowe używane do sterowania kierunkiem pracy silników
int m2direction=4;
boolean crash=false;

void setup()
{
   pinMode(m1direction, OUTPUT);
   pinMode(m2direction, OUTPUT);
   attachInterrupt(0, backOut, RISING);
   delay(5000);
}

❶ void backOut()
{
   crash=true;
}

❷ void backUp()
{
   digitalWrite(m1direction,LOW); // ruch w tył
   digitalWrite(m2direction,LOW); // ruch w tył
   analogWrite(m1speed, 200); // szybkość
   analogWrite(m2speed, 200);
   delay(2000);
   analogWrite(m1speed, 0); // szybkość
   analogWrite(m2speed, 0);
}
```

```
void goForward(int duration, int pwm)
{
   long a,b;
   boolean move=true;
❸  a=millis();
   do
   {
      if (crash==false)
      {
         digitalWrite(m1direction,HIGH); // ruch w przód
         digitalWrite(m2direction,HIGH); // ruch w przód
         analogWrite(m1speed, pwm); // szybkość
         analogWrite(m2speed, pwm);
      }
      if (crash==true)
      {
         backUp();
         crash=false;
      }
❹     b=millis()-a;
      if (b>=duration)
      {
         move=false;
      }
   } while (move!=false);
   // zatrzymuje silniki
   analogWrite(m1speed, 0);
   analogWrite(m2speed, 0);
}

void loop()
{
   goForward(5000, 255);
   delay(2000);
}
```

W tym szkicu zastosowano zaawansowaną metodę sterowania ruchem robota — do monitorowania ruchu tego pojazdu są używane dwie zmienne logiczne. Pierwszą z tych zmiennych jest `crash`. Jeśli robot uderzy w jakiś obiekt, powodując aktywację mikroprzełącznika, nastąpi wywołanie przerwania, które z kolei wywoła funkcję `backOut()` w wierszu ❶. Działanie tej zmiennej sprowadza się do zmiany wartości zmiennej `crash` z `false` na `true`. Druga monitorowania zmienna logiczna została nazwana `move`. W funkcji `goForward()` wykorzystano funkcję `millis()` w wierszu ❸, aby na bieżąco sprawdzać, czy robot nie zakończył ruchu zaplanowanego na określony okres (reprezentowany przez parametr `duration`).

W wierszu ❹ funkcja `goForward()` sprawdza, czy czas od rozpoczęcia ruchu jest krótszy od planowanego czasu tej operacji — jeśli tak, zmiennej `move` jest przypisywana wartość `true`. Oznacza to, że robot może się poruszać w przód,

pod warunkiem że nie uderzył w żaden obiekt i nie zakończył się czas zaplanowany na bieżący ruch. W razie wykrycia kolizji następuje wywołanie funkcji `backUp()` zdefiniowanej w wierszu ❷, która powoduje powolne wycofywanie robota przez 2 sekundy.

UWAGA *Warto zastosować w tym szkicu także alternatywne funkcje ruchu z projektu nr 40, aby rozbudować lub zmodyfikować ten przykład.*

Czujniki odległości na podczerwień

Następna metoda unikania kolizji, którą zastosujemy, polega na zastosowaniu czujnika odległości na podczerwień (IR). Czujnik wysyła sygnał w formie promieniowania podczerwonego i zwraca napięcie proporcjonalne do odległości pomiędzy źródłem tego promieniowania a powierzchnią, od której zostało odbite. Czujniki podczerwieni są przydatne podczas wykrywania kolizji przede wszystkim z uwagi na niską cenę; warto jednak pamiętać, że nie nadają się do precyzyjnego mierzenia odległości. W tym projekcie zostanie użyty czujnik analogowy Sharp GP2Y0A21YK0F (patrz rysunek 12.23).

Rysunek 12.23. Czujnik podczerwieni firmy Sharp

Łączenie obwodu

Aby podłączyć czujnik, należy podłączyć czerwony, czarny i biały przewód odpowiednio do pinów 5 V i GND oraz analogowego pinu wejściowego na płytce Arduino. Do mierzenia napięcia zwracanego przez ten czujnik będziemy używali funkcji `analogRead()`. Wykres pokazany na rysunku 12.24 ilustruje relacje pomiędzy zmierzoną odległością a napięciem wyjściowym czujnika.

Testowanie czujnika odległości na podczerwień

Ponieważ zależność odległości i napięcia wyjściowego trudno wyrazić w formie równania matematycznego, odczyty zostaną podzielone na przedziały odpowiadające 5 cm odcinkom. Do zademonstrowania działania czujnika użyjemy prostego przykładu. Wystarczy podłączyć biały przewód czujnika podczerwieni do pinu analogowego nr 0, czerwony przewód do pinu 5 V i czarny przewód do pinu GND, a następnie wpisać i wysłać na płytkę Arduino szkic z listingu 12.2.

Rysunek 12.24. Wykres zależności napięcia wyjściowego od odległości do obiektu odbijającego promieniowanie

Listing 12.2. Szkic demonstrujący działanie czujnika podczerwieni

```
// Listing 12.2

float sensor = 0;
int cm = 0;

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   sensor = analogRead(0);
   if (sensor<=90)
   {
      Serial.println("Nieskończona odległość!");
   } else if (sensor<100) // 80 cm
   {
      cm = 80;
   } else if (sensor<110) // 70 cm
   {
      cm = 70;
   } else if (sensor<118) // 60 cm
```

```
    {
       cm = 60;
    } else if (sensor<147) // 50 cm
    {
       cm = 50;
    } else if (sensor<188) // 40 cm
    {
       cm = 40;
    } else if (sensor<230) // 30 cm
    {
       cm = 30;
    } else if (sensor<302) // 25 cm
    {
       cm = 25;
    } else if (sensor<360) // 20 cm
    {
       cm = 20;
    } else if (sensor<505) // 15 cm
    {
       cm = 15;
    } else if (sensor<510) // 10 cm
    {
       cm = 10;
    } else if (sensor>=510) // Za blisko!
    {
       Serial.println("Za blisko!");
    }
    Serial.print("Odległość: ");
    Serial.print(cm);
    Serial.println(" cm");
    delay(250);
}
```

Szkic odczytuje napięcie wyjściowe czujnika podczerwieni w wierszu ❶, a następnie określa przybliżoną odległość do obiektu za pomocą sekwencji wyrażeń `if` (począwszy od wiersza ❷). Odległość jest określana na podstawie napięcia zwróconego przez czujnik przy użyciu dwóch parametrów. Pierwszym parametrem jest relacja napięcia do odległości zgodnie z wykresem widocznym na rysunku 12.24. Jak napisałem przy okazji omawiania projektu nr 6, funkcja `analogRead()` zwraca wartość z przedziału od 0 do 1023 reprezentującą odpowiednio napięcie od 0 do około 5 V. Oznacza to, że na podstawie tej wartości możemy obliczyć przybliżoną odległość wskazaną przez czujnik.

Po wysłaniu szkicu należy otworzyć okno monitora portu szeregowego i sprawdzić działanie czujnika, trzymając dłoń lub kartkę papieru w różnej odległości od czujnika. W oknie monitora portu szeregowego powinny pojawić się przybliżone odległości (patrz rysunek 12.25).

Rysunek 12.25. Wyniki wyświetlone przez szkic z listingu 12.2

Projekt nr 42: wykrywanie kolizji robota za pomocą czujnika odległości na podczerwień

Spróbujmy teraz wykorzystać czujnik podczerwieni do wykrywania kolizji robota (zamiast stosowanego wcześniej mikroprzełącznika). W tym projekcie nieznacznie zmodyfikujemy szkic z projektu nr 41. Zamiast stosowania przerwań w szkicu zostanie utworzona funkcja `checkDistance()`, która zmieni wartość zmiennej `crash` na `true` w sytuacji, gdy czujnik podczerwieni wykryje, że odległość od przeszkody jest mniejsza lub równa 20 cm. Funkcja będzie wywoływana w ciele funkcji `goForward()` (odpowiedzialnej za ruch w przód) w ramach pętli `do-while`.

Po podłączeniu czujnika podczerwieni do płytki Arduino robota należy wpisać i wysłać następujący szkic:

```
// Projekt nr 42 — wykrywanie kolizji robota za pomocą czujnika odległości na podczerwień

int m1speed=6; // piny cyfrowe używane do sterowania szybkością
int m2speed=5;
int m1direction=7; // piny cyfrowe używane do sterowania kierunkiem pracy silników
int m2direction=4;
boolean crash=false;

void setup()
{
   pinMode(m1direction, OUTPUT);
   pinMode(m2direction, OUTPUT);
   delay(5000);
}
```

```
void backUp()
{
   digitalWrite(m1direction,LOW); // ruch w tył
   digitalWrite(m2direction,LOW); // ruch w tył
   analogWrite(m1speed, 200); // szybkość
   analogWrite(m2speed, 200);
   delay(2000);
   analogWrite(m1speed, 0); // szybkość
   analogWrite(m2speed, 0);
}

void checkDistance()
{
❶ if (analogRead(0)>460)
   {
      crash=true;
   }
}

void goForward(int duration, int pwm)
{
   long a,b;
   boolean move=true;
   a=millis();
   do
   {
      checkDistance();
      if (crash==false)
      {
         digitalWrite(m1direction,HIGH); // ruch w przód
         digitalWrite(m2direction,HIGH); // ruch w przód
         analogWrite(m1speed, pwm); // szybkość
         analogWrite(m2speed, pwm);
      }
      if (crash==true)
      {
         backUp();
         crash=false;
      }
      b=millis()-a;
      if (b>=duration)
      {
         move=false;
      }
   } while (move!=false);
   // zatrzymuje silniki
   analogWrite(m1speed, 0);
   analogWrite(m2speed, 0);
}
```

```
void loop()
{
   goForward(5000, 255);
   delay(2000);
}
```

W szkicu zastosowano te same metody co w projekcie nr 41 z tą różnicą, że tym razem odległość jest stale mierzona w wierszu ❶, a zmiennej `crash` jest przypisywana wartość `true`, jeśli odległość pomiędzy czujnikiem podczerwieni a obiektem na drodze robota jest mniejsza niż 20 cm.

Po uruchomieniu robota wyposażonego w ten czujnik szybko docenimy zalety czujnika kolizji, który nie wymaga kontaktu z obiektem przed robotem. Dobrym rozwiązaniem byłoby zamontowanie dodatkowych czujników, na przykład także na tyle robota i na wszystkich narożnikach. Dodanie kodu sprawdzającego kolejno każdy z tych czujników i podejmującego decyzję na podstawie zwróconych wartości (zmierzonych odległości) nie powinno nam sprawić żadnego kłopotu.

Ultradźwiękowe czujniki odległości

Ostatnia metoda unikania kolizji, którą warto zastosować w budowanym robocie, polega na użyciu *ultradźwiękowego czujnika odległości*. Czujnik wysyła fale dźwiękowe o częstotliwości niesłyszalnej dla ludzkiego ucha i mierzy czas, który upłynął od momentu wysłania sygnału do jego powrotu (po odbiciu od przeszkody). W tym projekcie zostanie użyty ultradźwiękowy czujnik odległości Parallax Ping))) (patrz rysunek 12.26), który jest nie tylko niedrogi, ale też umożliwia mierzenie odległości z dokładnością do 1 cm.

Rysunek 12.26. Ultradźwiękowy czujnik odległości Ping)))

Czujniki ultradźwiękowe mogą precyzyjnie mierzyć odległość od obiektów oddalonych o 2 do 300 cm. Ponieważ jednak odbita fala dźwiękowa musi wrócić do czujnika, czujnik nie może być ustawiony pod kątem większym niż 45 stopni względem kierunku ruchu robota.

Łączenie czujnika ultradźwiękowego

Aby połączyć czujnik z płytką Arduino, należy połączyć końcówki 5 V i GND z odpowiednimi pinami oraz końcówkę SIG (od ang. *signal*) z dowolnym pinem cyfrowym.

Stosowanie czujnika ultradźwiękowego

Czujnik ultradźwiękowy mierzy odległość tylko po otrzymaniu odpowiedniego żądania. Aby zmierzyć odległość, należy wysłać bardzo krótki sygnał `HIGH` (trwający zaledwie 5 mikrosekund) do pinu SIG. Po chwili czujnik powinien zwrócić sygnał `HIGH`, którego długość reprezentuje czas oczekiwania na powrót fali ultradźwiękowej wysłanej wcześniej przez ten czujnik. Wartość należy podzielić przez 2, aby określić faktyczną odległość dzielącą czujnik od obiektu, od którego został odbity sygnał dźwiękowy.

Użycie czujnika wymaga wykorzystania tego samego pinu cyfrowego do wysyłania i odbierania danych oraz opracowania dwóch nowych funkcji:

- funkcji **`delayMicroseconds(mS)`** wstrzymującej wykonywanie szkicu systemu Arduino na czas określony za pomocą parametru mS (w mikrosekundach);
- funkcji **`pulseDuration(pin, HIGH)`** mierzącej długość impulsu `HIGH` na pinie cyfrowym i zwracającej ten czas (wyrażony w mikrosekundach).

Po uzyskaniu czasu trwania przychodzącego impulsu można przekonwertować tę wartość na centymetry, dzieląc liczbę mikrosekund przez 29,412 (ponieważ sygnał dźwiękowy pokonuje 340 metrów w ciągu sekundy, czyli 34 centymetry w ciągu milisekundy).

Testowanie ultradźwiękowego czujnika odległości

Aby uprościć stosowanie czujnika ultradźwiękowego, posłużymy się funkcją `getDistance()` z listingu 12.3. Pin SIG czujnika ultradźwiękowego należy połączyć z pinem cyfrowym nr 3, a następnie wpisać i wysłać na płytkę Arduino następujący szkic:

Listing 12.3. Szkic demonstrujący działanie czujnika ultradźwiękowego

```
// Listing 12.3

int signal=3;

void setup()
{
   pinMode(signal, OUTPUT);
   Serial.begin(9600);
}
```

```
int getDistance()
// zwraca odległość (w centymetrach) wykrytą przez czujnik Ping)))
{
   int distance;
   unsigned long pulseduration=0;

   // pobiera pomiar czujnika odległości Ping)))
   // ustawia pin wyjściowy potrzebny do wysłania impulsu
❶  pinMode(signal, OUTPUT);

   // ustawia stan LOW na tym pinie
   digitalWrite(signal, LOW);
   delayMicroseconds(5);

❷  // wysyła impuls trwający 5 mikrosekund, aby aktywować czujnik Ping)))
   digitalWrite(signal, HIGH);
   delayMicroseconds(5);
   digitalWrite(signal, LOW);

❸  // zmienia tryb pracy pinu cyfrowego na wejście, aby odczytać impuls przychodzący
   pinMode(signal, INPUT);

   // mierzy długość impulsu przychodzącego
   pulseduration=pulseIn(signal, HIGH);

❹  // dzieli długość impulsu przez dwa
   pulseduration=pulseduration/2;

❺  // konwertuje czas impulsu na centymetry
   distance = int(pulseduration/29);
   return distance;
}

void loop()
{
   Serial.print(getDistance());
   Serial.println(" cm");
   delay(500);
}
```

Funkcja `int getDistance()` zwraca odległość dzielącą czujnik od przeszkody. Wiersze od ❶ do ❺ dobrze ilustrują procedurę wysyłania impulsu do czujnika i mierzenia czasu sygnału zwróconego przez ten czujnik — na tej podstawie funkcja oblicza odległość.

Po wysłaniu szkicu na płytkę Arduino należy otworzyć okno monitora portu szeregowego i sprawdzić działanie czujnika, umieszczając jakiś obiekt w różnych odległościach od czujnika. W oknie monitora portu szeregowego powinny pojawić się odległości do tego obiektu wyrażone w centymetrach (patrz rysunek 12.27).

Rysunek 12.27. Wyniki wyświetlone przez szkic z listingu 12.3

Projekt nr 43: wykrywanie kolizji robota za pomocą ultradźwiękowego czujnika odległości

Skoro rozumiemy już działanie tego czujnika, spróbujmy zastosować ten komponent w robocie.

Szkic

W tym projekcie do testowania zbliżającej się kolizji zostanie użyta funkcja `get ↪Distance()` z listingu 12.3. W poniższym szkicu sprawdzamy, czy odległość od obiektu na drodze robota jest mniejsza niż 10 cm — jeśli tak, robot wycofuje się. W środowisku IDE należy wpisać i wysłać na płytkę Arduino następujący szkic, aby sprawdzić działanie nowego czujnika:

```
// Projekt nr 43 — wykrywanie kolizji robota za pomocą ultradźwiękowego
// czujnika odległości

int m1speed=6; // piny cyfrowe używane do sterowania szybkością
int m2speed=5;
int m1direction=7; // piny cyfrowe używane do sterowania kierunkiem pracy silników
int m2direction=4;
int signal=3;
boolean crash=false;

void setup()
{
   pinMode(m1direction, OUTPUT);
   pinMode(m2direction, OUTPUT);
   pinMode(signal, OUTPUT);
   delay(5000);
   Serial.begin(9600);
```

```
}

int getDistance()
// zwraca odległość (w centymetrach) wykrytą przez czujnik Ping)))
{
   int distance;
   unsigned long pulseduration=0;

   // pobiera pomiar czujnika odległości Ping)))
   // ustawia pin wyjściowy potrzebny do wysłania impulsu
   pinMode(signal, OUTPUT);

   // ustawia stan LOW na tym pinie
   digitalWrite(signal, LOW);
   delayMicroseconds(5);

   // wysyła impuls trwający 5 mikrosekund, aby aktywować czujnik Ping)))
   digitalWrite(signal, HIGH);
   delayMicroseconds(5);
   digitalWrite(signal, LOW);

   // zmienia tryb pracy pinu cyfrowego na wejście, aby odczytać impuls przychodzący
   pinMode(signal, INPUT);

   // mierzy długość impulsu przychodzącego
   pulseduration=pulseIn(signal, HIGH);

   // dzieli długość impulsu przez dwa
   pulseduration=pulseduration/2;

   // konwertuje czas impulsu na centymetry
   distance = int(pulseduration/29);
   return distance;
}

void backUp()
{
   digitalWrite(m1direction,LOW); // ruch w tył
   digitalWrite(m2direction,LOW);
   delay(2000);
   digitalWrite(m1direction,HIGH); // ruch w lewo
   digitalWrite(m2direction,LOW);
   analogWrite(m1speed, 200); // szybkość
   analogWrite(m2speed, 200);
   delay(2000);
   analogWrite(m1speed, 0); // szybkość
   analogWrite(m2speed, 0);
}

void goForward(int duration, int pwm)
```

```
{
   long a,b;
   int dist=0;
   boolean move=true;
   a=millis();
   do
   {
      dist=getDistance();
      Serial.println(dist);
❶     if (dist<10) // jeśli odległość do obiektu jest mniejsza niż 10 cm
      {
         crash=true;
      }
      if (crash==false)
      {
         digitalWrite(m1direction,HIGH); // ruch w przód
         digitalWrite(m2direction,HIGH); // ruch w przód
         analogWrite(m1speed, pwm); // szybkość
         analogWrite(m2speed, pwm);
      }
      if (crash==true)
      {
         backUp();
         crash=false;
      }
      b=millis()-a;
      if (b>=duration)
      {
         move=false;
      }
   } while (move!=false);
   // zatrzymuje silniki
   analogWrite(m1speed, 0);
   analogWrite(m2speed, 0);
}

void loop()
{
   goForward(1000, 255);
}
```

Także w tym przypadku odległość od ewentualnej przeszkody jest stale mierzona w wierszu ❶; jeśli odległość sygnalizowana przez czujnik ultradźwiękowy jest mniejsza niż 10 cm, zmiennej `crash` jest przypisywana wartość `true`. Obserwowanie robota, który w magiczny sposób unika kolizji z innymi obiektami lub toczy boje na inteligencję ze zwierzętami domowymi, może być niezwykle ciekawe.

Co dalej?

W tym rozdziale pokazałem, jak wprowadzić projekty na bazie platformy Arduino w świat ruchu. Za pomocą prostych silników (lub pary silników) i odpowiedniego modułu dodatkowego można budować urządzenia zdolne do samodzielnego ruchu, a nawet omijania przeszkód. Wykorzystaliśmy trzy typy czujników, aby zademonstrować zastosowania komponentów oferujących różne poziomy dokładności (dostępne w różnych cenach), tak by każdy konstruktor sam podjął decyzję z uwzględnieniem wymagań projektu i planowanego budżetu.

Mam nadzieję, że perspektywa samodzielnego projektowania i budowy podobnych konstrukcji jest wystarczająco kusząca. Na tym jednak nie koniec. W następnym rozdziale wyjdziemy na świeże powietrze i wykorzystamy potencjał nawigacji satelitarnej.

13

Stosowanie systemu GPS na platformie Arduino

W tym rozdziale:

- nauczysz się, jak łączyć moduł GPS z płytką Arduino;
- zbudujesz proste urządzenie wyświetlające współrzędne geograficzne;
- nauczysz się wyświetlać aktualne położenie na mapie;
- zbudujesz precyzyjny zegar;
- zarejestrujesz położenie ruchomego obiektu w czasie.

Dowiesz się, jak za pomocą niedrogiego modułu GPS określać bieżącą pozycję, skonstruować precyzyjny zegar oraz zbudować urządzenie rejestrujące (na karcie microSD) położenie w dłuższym czasie, dzięki czemu będzie można nanieść zapisaną historię ruchów na mapę.

Czym jest GPS?

GPS (od ang. *global positioning system*) to system nawigacji satelitarnej, w którym satelity na orbitach okołoziemskich wysyłają sygnały do urządzeń na Ziemi. Na podstawie tych sygnałów urządzenia mogą określać swoje położenie i bieżącą godzinę w każdym zakątku globu. Większość z nas miała do czynienia z urządzeniami nawigacji satelitarnej korzystającymi z tego systemu w samochodach i telefonach komórkowych.

Mimo że budowa na bazie platformy Arduino urządzenia do nawigacji, które będzie wyświetlało szczegółową mapę, jest niemożliwa, możemy użyć modułu GPS do określania położenia, godziny i przybliżonej szybkości (jeśli urządzenie jest w ruchu). Wykorzystamy do tego celu moduł odbiornika GPS nazwany EM406 (patrz rysunek 13.1).

Rysunek 13.1. Odbiornik GPS EM406

Stosowanie tego odbiornika wymaga użycia dodatkowej płytki GPS dla platformy Arduino (cały zestaw jest dostępny w sklepie SparkFun — nr produktu: RTL-10709). Zestaw pokazano na rysunku 13.2. Produkt obejmuje odbiornik, moduł dodatkowy dla platformy Arduino i przewód połączeniowy.

Rysunek 13.2. Kompletny zestaw modułu GPS

Warto też kupić 30-centymetrowy przewód (produkt nr GPS-09123 w sklepie SparkFun; patrz rysunek 13.3), który znacznie ułatwi instalację odbiornika GPS we właściwym miejscu.

Rysunek 13.3. Długi przewód do łączenia odbiornika z modułem GPS

Testowanie modułu GPS

Kupiony zestaw GPS warto sprawdzić, aby mieć pewność, że działa prawidłowo i odbiera sygnały wysyłane przez satelity systemu GPS. Odbiornik GPS musi „widzieć" niebo, ale sygnał GPS przechodzi przez okna. Zwykle najlepszym miejscem dla testów jest wolne powietrze, jednak w wielu przypadkach wystarczy umieszczenie odbiornika przy oknie lub świetliku. Aby przetestować działanie odbiornika, należy przygotować moduł dla platformy Arduino i uruchomić prosty szkic wyświetlający odbierane dane.

W pierwszym kroku trzeba podłączyć odbiornik GPS do modułu oraz zamontować ten moduł na płytce Arduino. Warto zwrócić uwagę na niewielki przełącznik na module GPS (patrz rysunek 13.4).

Rysunek 13.4. Przełącznik danych na module GPS

Przed wysłaniem i uruchomieniem szkicu korzystającego z modułu GPS należy ustawić przełącznik w pozycji DLINE, a następnie w pozycji UART. Warto też sprawdzić, czy jest włączony przełącznik zasilania odbiornika.

W środowisku Arduino IDE należy wpisać i wysłać na płytkę szkic z listingu 13.1.

Listing 13.1. Prosty szkic testowy

```
// Listing 13.1

void setup()
{
❶ Serial.begin(4800);
}

void loop()
{
   byte a;
❷ if ( Serial.available() > 0 )
   {
      a = Serial.read(); // pobiera bajt danych z modułu GPS
❸    Serial.write(a);
   }
}
```

Szkic nasłuchuje danych na porcie szeregowym w wierszu ❷. Po otrzymaniu bajta danych od modułu GPS szkic wysyła te dane w nieprzetworzonej formie do monitora portu szeregowego w wierszu ❸. (Warto zwrócić uwagę na szybkość portu szeregowego na poziomie 4800 bitów na sekundę ustawioną w wierszu ❶ — zastosowana wartość odpowiada szybkości przekazywania danych przez odbiornik GPS).

Po wysłaniu szkicu należy ponownie ustawić przełącznik danych na pozycji UART. Warto też sprawdzić diodę LED reprezentującą stan odbiornika GPS. Jeśli dioda LED jest włączona, odbiornik GPS próbuje wykryć źródła sygnału. Po około 30 sekundach dioda LED powinna zacząć migać — oznacza to, że komponent odbiera dane z satelity. Zaraz po tym, jak dioda LED zacznie migać, należy otworzyć okno monitora portu szeregowego w środowisku Arduino IDE i ustawić szybkość odbierania danych na poziomie 4800. W oknie powinny być w sposób ciągły wyświetlane dane wynikowe podobne do tych z rysunku 13.5.

Rysunek 13.5. Nieprzetworzone dane odbierane z satelitów systemu GPS

Dane z odbiornika GPS są wysyłane do systemu Arduino po jednym znaku i na bieżąco są przekazywane do monitora portu szeregowego. Takie działanie odbiornika nie jest jednak zbyt praktyczne, zatem musimy użyć nowej biblioteki, która będzie wyodrębniała najważniejsze informacje z tych nieprzetworzonych danych i konwertowała je na bardziej przydatną formę. W tym celu należy pobrać i zainstalować bibliotekę *TinyGPS* dostępną na stronie *http://www.arduiniana.org/libraries/tinygps/* (zgodnie z metodą opisaną w podrozdziale „Rozszerzanie szkiców za pomocą bibliotek" w rozdziale 8.).

Projekt nr 44: budowa prostego odbiornika GPS

Spróbujmy teraz skonstruować prosty odbiornik GPS. Ponieważ jednak odbiorniki GPS zwykle stosuje się na zewnątrz, z dala od komputera, musimy jeszcze uzupełnić obwód o moduł LCD, na którym będą wyświetlane dane (patrz rysunek 13.6).

Rysunek 13.6. Moduł LCD i klawiatury numerycznej firmy Freetronics

UWAGA *W tym i kolejnych projektach będzie używany moduł Freetronics LCD & Keypad. Więcej informacji na jego temat można znaleźć na stronie http://www.freetronics.com/products/lcd-keypad-shield. Czytelnicy, którzy zdecydują się zastosować inny moduł wyświetlacza, powinni we wszystkich szkicach zmienić wartości zdefiniowane w funkcji LiquidCrystal.*

Aby wyświetlać na module LCD współrzędne reprezentujące bieżącą pozycję, zbudujemy bardzo prosty przenośny odbiornik GPS zasilany baterią 9 V z odpowiednim złączem.

Sprzęt

Wymagania sprzętowe tego projektu są minimalne:

- płytka Arduino i przewód USB;
- moduł LCD lub wspomniany wcześniej moduł firmy Freetronics;

- bateria 9 V do zasilania gniazda DC;
- zestaw modułu GPS firmy SparkFun.

Szkic

W środowisku IDE należy wpisać i wysłać następujący szkic:

```
// Projekt nr 44 — budowa prostego odbiornika GPS
❶ #include <TinyGPS.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );

// Tworzy obiekt typu TinyGPS:
TinyGPS gps;

❷ void getgps(TinyGPS &gps);

void setup()
{
   Serial.begin(4800);
   lcd.begin(16, 2);
}

void getgps(TinyGPS &gps)
// Funkcja getgps wyświetla żądane dane na ekranie LCD
{
   float latitude, longitude;
   // dekoduje i wyświetla bieżącą pozycję
❸  gps.f_get_position(&latitude, &longitude);
   lcd.setCursor(0,0);
   lcd.print("Szer:");
   lcd.print(latitude,5);
   lcd.print(" ");
   lcd.setCursor(0,1);
   lcd.print("Dług:");
   lcd.print(longitude,5);
   lcd.print(" ");
   delay(3000); // czeka 3 sekundy
   lcd.clear();
}

void loop()
{
   byte a;
   if ( Serial.available() > 0 ) // jeśli otrzymano dane za pośrednictwem portu
                                 // szeregowego...
   {
      a = Serial.read();         // ...pobiera bajt danych
      if(gps.encode(a))          // jeśli otrzymano prawidłowe dane GPS...
      {
```

```
❹   getgps(gps);          // ...pobiera te dane i wyświetla je na
                          //    ekranie LCD
        }
    }
}
```

W wierszach od ❶ do ❷ szkic importuje i konfiguruje biblioteki niezbędne do obsługi modułów LCD i GPS. W ciele funkcji `void loop()`, w wierszu ❹, znaki odczytane z odbiornika GPS są przekazywane do funkcji `getgps()`, która w wierszu ❸ używa funkcji `gps.f_get_position()` do umieszczenia wartości reprezentujących pozycję w dwóch zmiennych typu `byte`: `&latitude` i `&longitude` (wartości tych zmiennych są następnie wyświetlane na ekranie LCD).

Wyświetlanie położenia na ekranie LCD

Po wysłaniu szkicu na platformę Arduino i po rozpoczęciu odbierania danych przez moduł GPS na ekranie LCD będą wyświetlane szerokość i długość geograficzna w formie liczb dziesiętnych (patrz rysunek 13.7).

Rysunek 13.7. Szerokość i długość geograficzna wyświetlone przez szkic z projektu nr 44

Co właściwie oznaczają te liczby? Dokładne miejsce reprezentowane przez te wartości można łatwo wskazać za pomocą Map Google (*https://maps.google.pl/*). Wystarczy wpisać w polu wyszukiwania tego serwisu szerokość i długość geograficzną oddzielone przecinkiem i spacją, a odpowiedni punkt zostanie wyświetlony na mapie. Na przykład dla współrzędnych widocznych na rysunku 13.7 zostanie wyświetlony punkt na mapie taki jak na rysunku 13.8.

Rysunek 13.8. Punkt na mapie dla współrzędnych widocznych na rysunku 13.7

Projekt nr 45: budowa precyzyjnego zegara korzystającego z systemu GPS

System GPS służy nie tylko do określania położenia — system dodatkowo wysyła dane o godzinie, dzięki czemu można łatwo skonstruować bardzo precyzyjny zegar.

Sprzęt

W tym projekcie zostanie użyty ten sam sprzęt co w projekcie nr 44.

Szkic

Aby zbudować zegar GPS, w środowisku Arduino IDE należy wpisać i wysłać na płytkę następujący szkic:

```
// Projekt nr 45 — budowa precyzyjnego zegara korzystającego z systemu GPS
#include <TinyGPS.h>
#include <LiquidCrystal.h>
LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );

// Tworzy obiekt typu TinyGPS:
TinyGPS gps;

void getgps(TinyGPS &gps);

void setup()
{
   Serial.begin(4800);
   lcd.begin(16, 2);
}

void getgps(TinyGPS &gps)
{
   int year,a,t;
   byte month, day, hour, minute, second, hundredths;
❶  gps.crack_datetime(&year,&month,&day,&hour,
   ↪&minute,&second,&hundredths);

❷  hour=hour+1; // należy poprawić zgodnie ze strefą czasową
   if (hour>23)
   {
      hour=hour-24;
   }
   lcd.setCursor(0,0); // wyświetla datę i godzinę
❸  lcd.print("Aktualna godzina ");
   lcd.setCursor(4,1);
   if (hour<10)
   {
```

```
      lcd.print("0");
   }
   lcd.print(hour, DEC);
   lcd.print(":");
   if (minute<10)
   {
      lcd.print("0");
   }
   lcd.print(minute, DEC);
   lcd.print(":");
   if (second<10)
   {
      lcd.print("0");
   }
   lcd.print(second, DEC);
}

void loop()
{
   byte a;
   if ( Serial.available() > 0 ) // jeśli otrzymano dane za pośrednictwem portu
                                 // szeregowego...
   {
      a = Serial.read();          // ...pobiera bajt danych
      if(gps.encode(a))           // jeśli otrzymano prawidłowe dane GPS...
      {
         getgps(gps);           // ...pobiera te dane i wyświetla je na ekranie LCD
      }
   }
}
```

Powyższy przykład działa bardzo podobnie do szkicu z projektu nr 44 z tą różnicą, że zamiast wyodrębniać dane o położeniu, kod wyodrębnia w wierszu ❶ aktualną godzinę (wskazywana godzina zawsze dotyczy strefy czasu uniwersalnego; ang. *Greenwich Mean Time* — UTC). W wierszu ❷ należy dodać lub odjąć odpowiednią liczbę godzin, aby uzyskać godzinę właściwą strefie czasowej, w której znajduje się urządzenie. Godzina jest formatowana i wyświetlana na ekranie LCD w wierszu ❸. Przykład wyświetlonej godziny pokazano na rysunku 13.9.

Rysunek 13.9. Efekt działania projektu nr 45

Projekt nr 46: rejestrowanie położenia ruchomego obiektu w czasie

Skoro wyjaśniłem już, jak odbierać współrzędne z systemu GPS i jak konwertować je na normalne zmienne, możemy wykorzystać tę wiedzę do budowy urządzenia rejestrującego położenie na karcie pamięci microSD (za pośrednictwem modułu kart opisanego w rozdziale 8.). Urządzenie będzie rejestrowało położenie w czasie — będzie sukcesywnie zapisywało dane uzyskiwane z systemu GPS. Moduł kart pamięci microSD umożliwia rejestrowanie ruchu samochodu, ciężarówki, łodzi lub dowolnego innego obiektu otrzymującego sygnał z systemu GPS, aby następnie wyświetlić zapisane informacje na komputerze.

Sprzęt

Do realizacji tego projektu będzie potrzebny ten sam sprzęt co w poprzednich przykładach z tą różnicą, że tym razem moduł LCD zostanie zastąpiony modułem kart pamięci microSD z rozdziału 8., a budowane urządzenie będzie zasilane baterią 9 V. W tym przykładzie będą rejestrowane godzina, informacje o położeniu oraz szacowana prędkość podróży.

Szkic

Po połączeniu wszystkich urządzeń należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 46 — rejestrowanie położenia ruchomego obiektu w czasie
#include <SD.h>
#include <TinyGPS.h>

// Tworzy obiekt typu TinyGPS:
TinyGPS gps;

void getgps(TinyGPS &gps);

void setup()
{
   pinMode(10, OUTPUT);
   Serial.begin(9600);
   // sprawdza, czy karta microSD jest dostępna i gotowa do użycia
   if (!SD.begin(8)) {
      Serial.println("Brak lub nieprawidłowe działanie karty");
      // zatrzymuje wykonywanie szkicu
      return;
   }
   Serial.println("Karta microSD jest gotowa");
}
```

```
void getgps(TinyGPS &gps)
{
   float latitude, longitude;
   int year;
   byte month, day, hour, minute, second, hundredths;

   // dekoduje i wyświetla bieżącą pozycję
   gps.f_get_position(&latitude, &longitude);
   File dataFile = SD.open("DATA.TXT", FILE_WRITE);
   // jeśli plik jest gotowy, zapisuje dane
❶  if (dataFile)
   {
❷     dataFile.print("Szer: ");
      dataFile.print(latitude,5);
      dataFile.print(" ");
      dataFile.print("Dług: ");
      dataFile.print(longitude,5);
      dataFile.print(" ");
      // dekoduje i wyświetla bieżącą datę i godzinę

gps.crack_datetime(&year,&month,&day,&hour,&minute,&second,&hundredths);
      // korekta strefy czasowej (tak jak w projekcie nr 45)
      hour=hour+1;
      if (hour>23)
      {
         hour=hour-24;
      }
      if (hour<10)
      {
         dataFile.print("0");
      }
      dataFile.print(hour, DEC);
      dataFile.print(":");
      if (minute<10)
      {
         dataFile.print("0");
      }
      dataFile.print(minute, DEC);
      dataFile.print(":");
      if (second<10)
      {
         dataFile.print("0");
      }
      dataFile.print(second, DEC);
      dataFile.print(" ");
      dataFile.print(gps.f_speed_kmph());
❸     dataFile.println(" km/h");
      dataFile.close();
❹     delay(30000); // rejestruje dane co 30 sekund
   }
```

```
}

void loop()
{
   byte a;
   if ( Serial.available() > 0 ) // jeśli otrzymano dane za pośrednictwem portu
                                 // szeregowego...
   {
      a = Serial.read(); // ...pobiera bajt danych
      if(gps.encode(a)) // jeśli otrzymano prawidłowe dane GPS...
      {
❺     getgps(gps); // ...pobiera te dane i wyświetla je na ekranie LCD
      }
   }
}
```

W tym szkicu zastosowano funkcję `void loop()` z projektów nr 44 i 45, aby pobierać dane z odbiornika GPS i przekazywać je do pozostałych funkcji programu. W wierszu ❺ tekst odczytany z odbiornika GPS jest przekazywany do biblioteki *TinyGPS* w celu dekodowania danych i przypisania wyodrębnionych informacji do odpowiednich zmiennych. W wierszu ❶ szkic sprawdza kartę microSD, aby określić, czy będzie możliwy zapis danych. W wierszach od ❷ do ❸ dane odczytane z odbiornika GPS są zapisywane w pliku tekstowym na karcie microSD. Ponieważ plik jest zamykany po każdym zapisie, użytkownik może bezpiecznie wyłączyć zasilanie płytki Arduino bez uprzedniego wyłączania szkicu (zasilanie należy wyłączyć przed umieszczeniem karty microSD w gnieździe i przed usunięciem jej z gniazda). I wreszcie w wierszu ❹ można zmienić okres dzielący kolejne operacje rejestrowania danych — wystarczy zmodyfikować wartość przekazywaną na wejściu funkcji `delay()`.

Wyświetlanie zarejestrowanych lokalizacji na mapie

Po uruchomieniu rejestratora pozycji GPS zawartość wygenerowanego pliku tekstowego powinna przypominać dane pokazane na rysunku 13.10.

Rysunek 13.10. Dane wynikowe wygenerowane przez projekt nr 46

Po zgromadzeniu danych można ręcznie wpisać kolejne współrzędne w serwisie Mapy Google, aby przeanalizować pokonaną drogę punkt po punkcie. Lepszym rozwiązaniem jest jednak naniesienie na mapę całej historii podróży. W tym celu należy otworzyć plik tekstowy w arkuszu kalkulacyjnym, wyodrębnić dane reprezentujące pozycje i dodać wiersz nagłówka (patrz rysunek 13.11). Plik należy następnie zapisać z rozszerzeniem *.csv*.

	A	B	C	D
1	długość	szerokość		
2	-37.86844	145.08743		
3	-37.86844	145.08743		
4	-37.86844	145.08743		
5	-37.86844	145.08743		
6	-37.86844	145.08743		
7	-37.86843	145.08743		
8	-37.86843	145.08741		
9	-37.86853	145.08732		
10	-37.86854	145.08731		
11	-37.86827	145.07974		
12	-37.86835	145.07955		

Rysunek 13.11. Dane o zarejestrowanych pozycjach

Należy teraz otworzyć stronę internetową GPS Visualizer (*http://www.gpsvisualizer.com/*). W ramce *Get Started Now* trzeba kliknąć przycisk *Przeglądaj* i wybrać plik danych. Z listy dostępnych formatów wyjściowych należy wybrać opcję *Google Maps*, a następnie nacisnąć przycisk *Go!*. Pozycje zarejestrowane przez rejestrator GPS powinny zostać wyświetlone na mapie podobnej do tej z rysunku 13.12 (mapę można przybliżać i przesuwać).

Rysunek 13.12. Dane rejestratora pozycji GPS naniesione na mapę

Co dalej?

Przykłady opisane w tym rozdziale pokazują, że czasem rozwiązania, które z początku wydają się zbyt skomplikowane, jak obsługa odbiorników GPS, w rzeczywistości są zadziwiająco proste, jeśli tylko zostaną zbudowane na bazie platformy Arduino. W następnym rozdziale omówię inne takie rozwiązania — sposoby tworzenia własnych połączeń bezprzewodowych i zdalnego sterowania urządzeniami.

14

Bezprzewodowe przesyłanie danych

W tym rozdziale wyjaśnię, jak wysyłać i odbierać instrukcje oraz dane za pośrednictwem różnych typów urządzeń do transmisji bezprzewodowej. W szczególności pokażę, jak:

- wysyłać cyfrowe sygnały wyjściowe za pośrednictwem tanich modułów komunikacji bezprzewodowej;
- utworzyć prosty i niedrogi system bezprzewodowego zdalnego sterowania;
- używać odbiorników i nadajników bezprzewodowych w technologii XBee;
- utworzyć zdalnie sterowany czujnik temperatury.

Stosowanie niedrogich modułów komunikacji bezprzewodowej

Wysyłanie danych tekstowych w jednym kierunku pomiędzy dwoma systemami Arduino za pośrednictwem połączenia bezprzewodowego jest dość łatwe — wystarczy użyć niedrogich modułów radiowych (RF), na przykład modułu nadajnika pokazanego na rysunku 14.1. Moduły tego typu zwykle są oferowane w parach nazywanych modułami lub zestawami *połączeń radiowych*. Dobrymi przykładami takich zestawów są produkty WLS107B4B firmy Seeed Studio oraz WRL-10534 firmy SparkFun. W projektach opisywanych w tym rozdziale będą używane także cieszące się największą popularnością moduły pracujące w paśmie fal radiowych 433 MHz.

Rysunek 14.1. Moduł nadajnika RF

Piny widoczne na dole modułu z rysunku 14.1 służą odpowiednio (od lewej strony) do podłączenia zasilania 5 V, wejściowego przewodu sygnałowego i zewnętrznej anteny. Antena może mieć postać zwykłego przewodu, a w przypadku transmisji danych na niewielkie odległości w ogóle nie jest potrzebna. (Każdy model modułu może być nieco inny, zatem przed przystąpieniem do łączenia obwodu warto sprawdzić dokumentację z opisem przeznaczenia poszczególnych pinów).

Na rysunku 14.2 pokazano moduł odbiornika, który jest nieznacznie większy od modułu nadajnika.

Rysunek 14.2. Moduł odbiornika RF

Podłączenie odbiornika jest bardzo proste — piny V+ i V− należy połączyć odpowiednio z pinami 5 V i GND na płytce Arduino, a pin DATA należy połączyć z pinem używanym do odbierania danych.

Zanim będzie możliwe korzystanie z opisanych modułów, należy jeszcze pobrać i zainstalować najnowszą wersję biblioteki *VirtualWire* ze strony *http://www.airspayce.com/mikem/arduino/*. Po zainstalowaniu tej biblioteki można przejść do realizacji projektu opisanego w następnym podrozdziale.

UWAGA *Moduły komunikacji bezprzewodowej RF są co prawda tanie i łatwe w użyciu, ale nie oferują mechanizmów sprawdzania błędów, które pozwoliłyby weryfikować poprawność odbieranych danych. Właśnie z uwagi na brak możliwości sprawdzania poprawności przesyłanych danych rekomenduję stosowanie modułów tego typu tylko podczas realizacji najprostszych zadań, na przykład do zdalnego sterowania urządzeniami. Jeśli projekt wymaga bardziej precyzyjnej lub niezawodnej formy transmisji danych, należy rozważyć użycie na przykład modułów XBee, które zostaną omówione w dalszej części tego rozdziału.*

Projekt nr 47: zdalne, bezprzewodowe sterowanie urządzeniem

W tym projekcie wykorzystamy łączność radiową do sterowania dwoma systemami Arduino — naciśnięcie przycisku podłączonego do jednej płytki spowoduje wysłanie sygnału za pośrednictwem cyfrowego pinu wyjściowego na innej, oddalonej płytce. Projekt będzie ilustrował sposoby stosowania modułów RF i pozwoli sprawdzić odległość niezawodnej współpracy najprostszych modułów komunikacji radiowej. Powyżej tej odległości trzeba będzie zastosować moduły zaprojektowane z myślą o bardziej zaawansowanych zadaniach. (W otwartym terenie maksymalna odległość dzieląca nadajnik i odbiornik wynosi około 100 metrów, jednak w pomieszczeniach, gdzie wysyłane fale radiowe napotykają przeszkody, dystans dzielący oba komponenty jest wyraźnie mniejszy).

Sprzęt składający się na obwód nadajnika

Konstrukcja obwodu nadajnika wymaga następującego sprzętu:

- płytki Arduino i przewodu USB;
- baterii 9 V i przewodu zasilania DC (takiego jak w rozdziale 12.);
- modułu nadajnika RF działającego w paśmie 433 MHz (na przykład produkt WRL-10534 dostępny w sklepie SparkFun);
- dwóch rezystorów 10 kΩ (R1 i R2);
- dwóch kondensatorów 100 nF (C1 i C2);
- dwóch przycisków;
- płytki uniwersalnej.

Schemat nadajnika

Obwód nadajnika składa się z dwóch przycisków (z układem eliminującym skutki zjawiska drgania) podłączonych do pinów cyfrowych nr 2 i 3 oraz z modułu nadajnika połączonego z płytką Arduino zgodnie z wcześniejszym opisem (patrz rysunek 14.3).

Rysunek 14.3. Obwód nadajnika dla projektu nr 47

Sprzęt składający się na obwód odbiornika

Konstrukcja obwodu odbiornika wymaga następującego sprzętu:

- płytki Arduino i przewodu USB;
- baterii 9 V i przewodu zasilania DC (takiego jak w rozdziale 12.);
- modułu odbiornika RF działającego w paśmie 433 MHz (na przykład produkt WRL-10532 dostępny w sklepie SparkFun);
- płytki uniwersalnej;
- dwóch dowolnych diod LED;
- dwóch rezystorów 560 Ω (R1 i R2).

Schemat odbiornika

Obwód odbiornika składa się z dwóch diod LED podłączonych do pinów cyfrowych nr 6 i 7 oraz z modułu odbiornika RF, którego pin danych jest połączony z pinem cyfrowym nr 8 (patrz rysunek 14.4).

Rysunek 14.4. Obwód odbiornika dla projektu nr 47

Płytkę uniwersalną, diody LED, rezystory i moduł odbiornika można zastąpić gotowym modułem Freetronics 433 MHz dla płytki Arduino (patrz rysunek 14.5).

Rysunek 14.5. Moduł odbiornika Freetronics 433 MHz

Szkic nadajnika

Przeanalizujmy teraz szkic nadajnika. Należy wpisać poniższy szkic w środowisku IDE i wysłać kod na płytkę Arduino, do której podłączono obwód nadajnika:

```
// Projekt nr 47 — zdalne, bezprzewodowe sterowanie urządzeniem — szkic nadajnika

❶ #include <VirtualWire.h>
  uint8_t buf[VW_MAX_MESSAGE_LEN];
  uint8_t buflen = VW_MAX_MESSAGE_LEN;

❷ const char *on2 = "a";
  const char *off2 = "b";
  const char *on3 = "c";
  const char *off3 = "d";

  void setup()
  {
❸ vw_set_ptt_inverted(true);     // wymagane do działania modułów RF
     vw_setup(300);              // ustawia szybkość przesyłania danych
❹ vw_set_tx_pin(8);
     pinMode(2, INPUT);
     pinMode(3, INPUT);
  }

  void loop()
  {
❺ if (digitalRead(2)==HIGH)
     {
        vw_send((uint8_t *)on2, strlen(on2));  // wysyła dane w świat
        vw_wait_tx();                          // czeka chwilę
        delay(200);
     }
     if (digitalRead(2)==LOW)
     {
❻       vw_send((uint8_t *)off2, strlen(off2));
        vw_wait_tx();
        delay(200);
     }
     if (digitalRead(3)==HIGH)
     {
        vw_send((uint8_t *)on3, strlen(on3));
        vw_wait_tx();
        delay(200);
     }
     if (digitalRead(3)==LOW)
     {
        vw_send((uint8_t *)off3, strlen(off3));
        vw_wait_tx();
        delay(200);
     }
  }
```

Aby użyć modułów łączności radiowej, w szkicu zastosowano funkcje wirtualnego połączenia (w wierszach ❶ i ❸). W wierszu ❹ wskazano pin cyfrowy nr 8 jako ten, do którego podłączono pin danych modułu nadajnika i który będzie sterował szybkością transmisji danych. (Równie dobrze można by wykorzystać dowolny inny pin cyfrowy z wyjątkiem pinów nr 0 i 1, które zakłócałyby pracę portu szeregowego).

Szkic nadajnika odczytuje stan dwóch przycisków połączonych z pinami cyfrowymi nr 2 i 3, a następnie wysyła pojedynczy znak (wybierany zależnie od stanu przycisków) do modułu łączności radiowej. Jeśli na przykład przycisk połączony z pinem cyfrowym nr 2 znajduje się w stanie `HIGH`, system Arduino wysyła znak *a*; jeśli ten przycisk znajduje się w stanie `LOW`, system wysyła znak *b*. Wszystkie cztery stany zadeklarowano, począwszy od wiersza ❷.

Za wysłanie znaku odpowiada jedno z czterech wyrażeń `if`, począwszy od wiersza ❺, na przykład ciało wyrażenia `if-then` w wierszu ❻. Wysyłana zmienna jest wykorzystywana dwukrotnie; poniżej pokazano przykład użycia zmiennej `on2`:

```
vw_send((uint8_t *)on2, strlen(on2));
```

Funkcja `vw_send`, która wysyła wartość zmiennej `on2`, musi otrzymać także długość tej zmiennej (rozumianą jako liczba znaków), stąd konieczność użycia funkcji `strlen()`.

Szkic odbiornika

Czas przygotować szkic odbiornika. Należy wpisać poniższy szkic w środowisku IDE i wysłać kod na płytkę Arduino, do której podłączono obwód odbiornika:

```
// Projekt nr 47 — zdalne, bezprzewodowe sterowanie urządzeniem — szkic odbiornika
#include <VirtualWire.h>

uint8_t buf[VW_MAX_MESSAGE_LEN];
uint8_t buflen = VW_MAX_MESSAGE_LEN;

void setup()
{
❶    vw_set_ptt_inverted(true); // wymagane do działania modułów RF
     vw_setup(300);
❷    vw_set_rx_pin(8);
     vw_rx_start();
     pinMode(6, OUTPUT);
     pinMode(7, OUTPUT);
}

void loop()
{
❸    if (vw_get_message(buf, &buflen))
     {
```

```
❹     switch(buf[0])
         {
            case 'a':
               digitalWrite(6, HIGH);
               break;
            case 'b':
               digitalWrite(6, LOW);
               break;
            case 'c':
               digitalWrite(7, HIGH);
               break;
            case 'd':
               digitalWrite(7, LOW);
               break;
         }
      }
   }
```

Tak jak w przypadku szkicu nadajnika, zastosowano tutaj funkcje wirtualnego połączenia konfigurujące moduł odbiornika transmisji radiowej (w wierszu ❶), ustawiające szybkość przesyłania danych i wskazujące pin cyfrowy na płytce Arduino, do którego podłączono pin danych odbiornika (w wierszu ❷).

Po uruchomieniu tego szkicu znak wysłany przez obwód nadajnika trafia do modułu łączności radiowej i jest przekazywany do systemu Arduino. Funkcja `vw_get_message()` w wierszu ❸ otrzymuje znak odebrany przez system Arduino. Otrzymane dane są następnie interpretowane przez wyrażenie `switch-case` w wierszu ❹. Na przykład naciśnięcie przycisku S1 należącego do obwodu nadajnika spowoduje wysłanie znaku *a*. Znak jest przekazywany do nadajnika, który ustawia stan `HIGH` na pinie cyfrowym nr 6, powodując włączenie diody LED.

Ta prosta para obwodów może być dobrym punktem wyjścia dla bardziej złożonych systemów sterowania, w których różne kody znakowe będą interpretowane przez obwód odbiornika.

Moduły bezprzewodowego przesyłania danych XBee — większy zasięg i szybsza transmisja

Jeśli projekt wymaga bezprzewodowego wysyłania danych na większe odległości i z większą szybkością niż w przypadku podstawowych modułów komunikacji radiowej, warto rozważyć zastosowanie modułów danych XBee. Moduły odbierają i wysyłają dane szeregowe bezpośrednio do co najmniej jednego układu XBee i dalej do komputera lub komputerów. Istnieje wiele modeli modułów XBee, jednak w tej książce będą stosowane tylko moduły nadajników z linii Series 1 (patrz rysunek 14.6).

Rysunek 14.6. Typowy nadajnik XBee

Łączenie tego nadajnika z płytką Arduino jest bardzo proste dzięki modułowi XBee (patrz rysunek 14.7).

Rysunek 14.7. Nadajnik XBee połączony z płytką Arduino za pośrednictwem typowego modułu XBee

Do komunikacji z komputerem można użyć płytki XBee Explorer (przykład takiej płytki, a konkretnie produkt nr WRL-08687 w sklepie SparkFun, pokazano na rysunku 14.8).

Rysunek 14.8. Moduł XBee połączony z komputerem za pośrednictwem płytki Explorer i przewodu USB

Moduły XBee nie wymagają stosowania dodatkowych bibliotek — działają jak proste mosty danych szeregowych wysyłających i odbierających dane za pośrednictwem linii szeregowej systemu Arduino.

Projekt nr 48: transmisja danych za pomocą modułów XBee

Ten projekt zademonstruje proste metody przesyłania danych z platformy Arduino do komputera dysponującego modułem XBee (podłączonym za pośrednictwem płytki Explorer). Przygotowanie sprzętu dla tego projektu wymaga wykonania dwóch prostych kroków. Należy najpierw zamontować nadajnik XBee na module XBee. Moduł XBee trzeba następnie zainstalować na płytce Arduino.

Warto zwrócić uwagę na niewielki przełącznik na module XBee dla platformy Arduino (patrz rysunek 14.9).

Rysunek 14.9. Przełącznik danych na module XBee

Przełącznik jest taki sam jak przełącznik na module GPS, którego używaliśmy w rozdziale 13. Przełącznik danych określa, czy dane ze złącza USB na płytce Arduino lub nadajnika XBee mają być wysyłane do mikrokontrolera. Podczas wysyłania szkicu na platformę Arduino należy ustawić przełącznik w pozycji DLINE; przełącznik trzeba ponownie ustawić w pozycji UART podczas uruchamiania szkicu.

Szkic

W środowisku Arduino IDE należy wpisać i wysłać następujący szkic:

```
// Projekt nr 48 — transmisja danych za pomocą modułów XBee

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   Serial.println("Witaj świecie");
   delay(1000);
}
```

Mimo że działanie tego kodu sprowadza się do przekazania komunikatu „Witaj świecie” do portu szeregowego, szkic dobrze demonstruje łatwość, z jaką można wysyłać dane za pośrednictwem modułów XBee — wystarczy zapisać dane w linii szeregowej, a moduł XBee realizuje wszystkie niezbędne działania. Trzeba teraz usunąć przewód USB z płytki Arduino i podłączyć ją do zewnętrznego źródła zasilania, na przykład baterii 9 V i przewodu połączeniowego stosowanego w rozdziale 12. (Nie należy zapominać o ponownym ustawieniu przełącznika w pozycji UART).

Konfiguracja komputera pod kątem odbierania danych

Musimy jeszcze skonfigurować komputer tak, aby mógł otrzymywać dane. Za pośrednictwem płytki Explorer (pokazanej na rysunku 14.8) trzeba podłączyć nadajnik XBee do komputera. Następnie należy pobrać i zainstalować program Terminal dla systemu Windows. Program jest dostępny na stronie *https://sites.google.com/site/terminalbpp/*. Po otwarciu programu Terminal na ekranie zostanie wyświetlone okno podobne do tego z rysunku 14.10.

Rysunek 14.10. Terminal — oprogramowanie emulatora

W górnej części ekranu należy wybrać opcję *9600* w sekcji *Baud rate*, wartość *8* w sekcji *Data bits*, opcję *none* w sekcji *Parity*, wartość *1* w sekcji *Stop bits* i opcję *none* w sekcji *Handshaking*. Następnie należy kliknąć przycisk *ReScan* w lewym górnym rogu ekranu, aby wymusić na programie wybór właściwego portu USB. I wreszcie należy kliknąć przycisk *Connect*, a po chwili w oknie programu Terminal powinny zostać wyświetlone dane wysyłane przez system Arduino (patrz rysunek 14.11).

Rysunek 14.11. Odbieranie danych wysyłanych przez zdalny nadajnik XBee

Do jednego komputera można podłączyć wiele nadajników XBee, z których każdy może wysyłać własne dane w celu monitorowania wielu urządzeń. W ten sposób można w komfortowych warunkach, nie wstając od biurka, monitorować różne czujniki (na przykład czujniki temperatury) rozmieszczone na dużym obszarze. W podobny sposób można nawet sterować wieloma systemami Arduino z modułami XBee — przykładowe rozwiązanie tego typu skonstruujemy w następnym projekcie.

Projekt nr 49: budowa zdalnie sterowanego termometru

W tym projekcie zbudujemy zdalnie sterowany termometr, który będzie zwracał temperaturę po otrzymaniu odpowiedniego żądania wysłanego za pośrednictwem komputera. Podczas realizacji tego projektu wyjaśnię podstawy technik pobierania danych ze zdalnego czujnika — proponowane rozwiązanie będzie więc dobrym punktem wyjścia dla innych projektów wymagających stosowania zdalnych czujników.

Sprzęt

Projekt wymaga następujących urządzeń:

- płytki Arduino i przewodu USB;
- czujnika temperatury TMP36;
- dwóch nadajników XBee (Series 1);
- modułu dodatkowego XBee dla platformy Arduino;
- płytki XBee Explorer i przewodu USB;
- różnych przewodów;
- baterii 9 V i przewodu zasilania DC (takiego jak w rozdziale 12.);
- płytki uniwersalnej.

Układ urządzenia

Nadajnik XBee należy umieścić na module, a następnie zainstalować ten moduł na płytce Arduino. Czujnik TMP36 należy zainstalować na małej płytce uniwersalnej. Końcówki czujnika trzeba połączyć z pinami 5 V, GND i A0 na płytce modułu XBee (tak jak w projekcie nr 8; patrz rysunek 14.12).

Rysunek 14.12. Zdalny czujnik temperatury na bazie platformy Arduino

Płytkę USB Explorer z drugim nadajnikiem XBee należy podłączyć do komputera PC już po wysłaniu na płytkę Arduino opisanego poniżej szkicu.

Szkic

W środowisku IDE należy wpisać i wysłać następujący szkic:

```
// Projekt nr 49 — budowa zdalnie sterowanego termometru

char a;
float voltage=0;
float sensor=0;
float celsius=0;
float fahrenheit=0;
float photocell=0;

void setup()
{
   Serial.begin(9600);
}

void sendC()
{
   sensor=analogRead(0);
```

```
   voltage=((sensor*5000)/1024);
   voltage=voltage-500;
   celsius=voltage/10;
   fahrenheit=((celsius*1.8)+32);
   Serial.print("Temperatura: ");
   Serial.print(celsius,2);
   Serial.println(" stopni C");
}

void sendF()
{
   sensor=analogRead(0);
   voltage=((sensor*5000)/1024);
   voltage=voltage-500;
   celsius=voltage/10;
   fahrenheit=((celsius*1.8)+32);
   Serial.print("Temperatura: ");
   Serial.print(fahrenheit,2);
   Serial.println(" stopni F");
}

void getCommand()
❶ {
   Serial.flush();
   while (Serial.available() == 0)
   {
      // nie podejmuje żadnych działań do czasu otrzymania danych od modułu XBee
   }
   while (Serial.available() > 0)
   {
      a = Serial.read(); // odczytuje liczbę z bufora portu szeregowego
   }
}

void loop()
{
   getCommand(); // nasłuchuje poleceń wysłanych z komputera PC
❷    switch (a)
   {
❸    case 'c':
         // wysyła temperaturę w stopniach Celsjusza
         sendC();
         break;
❹        case 'f':
         // wysyła temperaturę w stopniach Fahrenheita
         sendF();
         break;
   }
}
```

Szkic nasłuchuje danych na porcie szeregowym za pomocą funkcji `getCommand()` zdefiniowanej w wierszu ❶. Funkcja zawiera pętlę, która nie podejmuje żadnych działań do momentu otrzymania jakiegoś znaku — funkcja zapisuje wówczas ten znak w zmiennej `a`. Po otrzymaniu znaku funkcja przystępuje do jego interpretacji za pomocą wyrażenia `switch-case` w wierszu ❷. W przypadku otrzymania litery *c* funkcja `sendC()`❸ oblicza temperaturę wyrażoną w stopniach Celsjusza, wysyła tę temperaturę do portu szeregowego, a następnie do komputera PC za pośrednictwem nadajników XBee. W przypadku otrzymania litery *f* funkcja `sendF()` ❹ oblicza temperaturę wyrażoną w stopniach Fahrenheita i wysyła ją do komputera PC.

Obsługa urządzenia

Po uruchomieniu programu Terminal (opisanego we wcześniejszej części tego rozdziału) należy wysłać literę *c* lub *f*. Po upływie około sekundy w oknie programu powinna zostać wyświetlona temperatura zwrócona przez zdalny czujnik (ekran programu powinien wówczas przypominać okno pokazane na rysunku 14.13).

Rysunek 14.13. Dane wynikowe wygenerowane przez projekt nr 49

Co dalej?

W tym rozdziale wykazałem, jak łatwe może być zdalne sterowanie wieloma systemami Arduino. W ten sposób można na przykład sterować cyfrowymi pinami wyjściowymi, wysyłając dane z terminala na komputerze PC lub z innej płytki Arduino. Sam system Arduino może też interpretować otrzymywane rozkazy zakodowane w formie znaków. Wiedza zawarta w tym rozdziale i we wcześniejszych rozdziałach w zupełności wystarczy do budowy zaawansowanych urządzeń.

Okazuje się jednak, że opisane tutaj techniki nie wyczerpują tematu bezprzewodowej transmisji danych, zatem zachęcam do dalszej lektury, w szczególności następnego rozdziału, w którym na bazie platformy Arduino zbudujemy prosty pilot do telewizora.

15

Zdalne sterowanie za pomocą podczerwieni

W tym rozdziale:

- skonstruujesz i przetestujesz odbiornik podczerwieni;
- nauczysz się zdalnie sterować cyfrowymi pinami wyjściowymi na płytce Arduino;
- dodasz system zdalnego sterowania do robota gąsienicowego zbudowanego w rozdziale 12.

W tym rozdziale dowiodę, że dodanie do platformy Arduino niedrogiego modułu odbiornika w zupełności wystarczy do zdalnego sterowania tym systemem za pomocą impulsów światła podczerwonego.

Czym jest podczerwień?

Mimo że większość z nas na co dzień korzysta z pilotów na podczerwień, niewiele osób wie, jak naprawdę działają te piloty. Sygnały podczerwieni (ang. *infrared — IR*) to tak naprawdę promienie światła o długości fal niewidocznej dla

ludzkiego oka. Właśnie dlatego nawet po naciśnięciu przycisku pilota nie widzimy, jak zapala się niewielka dioda LED odpowiedzialna za emisję światła podczerwonego.

Piloty na podczerwień zawierają co najmniej po jednej specjalnej diodzie LED emitującej światło w paśmie podczerwieni i używanej do przesyłania sygnałów do urządzenia docelowego. Naciśnięcie przycisku pilota powoduje wielokrotne włączanie i wyłączanie tej diody LED według unikatowego wzorca przypisanego do każdego przycisku. Sygnał trafia do specjalnego odbiornika podczerwieni i jest konwertowany kolejno na impulsy elektryczne i właściwe dane w układach elektronicznych urządzenia docelowego. Czytelnicy zainteresowani wspomnianymi wzorcami mogą łatwo sprawdzić, jak wyglądają sygnały emitowane przez diodę LED pilota, obserwując ją przez wizjer aparatu fotograficznego lub aparatu wbudowanego w telefon.

Przygotowanie do użycia podczerwieni

Przed przystąpieniem do dalszych kroków należy zainstalować bibliotekę obsługującą zdalne sterowanie za pomocą podczerwieni dla platformy Arduino. W tym celu wystarczy odwiedzić stronę *https://github.com/shirriff/Arduino-IRremote/*, pobrać niezbędne pliki i zainstalować je zgodnie z procedurą opisaną w podrozdziale „Rozszerzanie szkiców za pomocą bibliotek" w rozdziale 8.

Odbiornik podczerwieni

Następny krok polega na skonfigurowaniu odbiornika podczerwieni i sprawdzeniu, czy działa prawidłowo. Na rynku są dostępne zarówno niezależne odbiorniki podczerwieni (patrz rysunek 15.1), jak i gotowe, połączone moduły (patrz rysunek 15.2), których montaż jest nieco prostszy.

Rysunek 15.1. Odbiornik podczerwieni

Rysunek 15.2. Gotowy, połączony moduł odbiornika podczerwieni

Niezależny odbiornik podczerwieni, który pokazano na rysunku 15.1, to produkt Vishay TSOP4138 dostępny między innymi w sklepach Newark (kod produktu: 59K0287) i element14 (kod produktu: 1040743). Dolną nóżkę tego odbiornika (przy ustawieniu takim jak na rysunku) należy połączyć z pinem cyfrowym na płytce Arduino, środkową nóżkę z pinem GND, a górną nóżkę z pinem 5 V.

Na rysunku 15.2 pokazano gotowy moduł odbiornika IR z przewodem. Moduły tego typu są dostępne między innymi w ofercie DFRobot. Do najważniejszych zalet tych modułów należą gotowe połączenia z wyprowadzonymi przewodami i czytelne etykiety.

Niezależnie od wybranego modułu we wszystkich opisanych poniżej przykładach pin D (linia danych) będzie łączony z pinem cyfrowym nr 11 na płytce Arduino, pin VCC będzie łączony z pinem 5 V, a pin GND z pinem GND.

Pilot

I wreszcie będziemy potrzebowali pilota. Wykorzystamy pilot telewizyjny firmy Sony (podobny do tego z rysunku 15.3). Czytelnicy, którzy nie dysponują pilotem firmy Sony, mogą kupić jakiś niedrogi uniwersalny pilot — wystarczy zastosować kody obowiązujące w urządzeniach Sony. Niezbędne informacje można znaleźć w instrukcji dołączonej do pilota.

Rysunek 15.3. Typowy pilot do telewizora firmy Sony

Szkic testowy

Warto teraz upewnić się, że wszystko działa prawidłowo. Po podłączeniu odbiornika podczerwieni do płytki Arduino należy wpisać i wysłać na tę płytkę szkic z listingu 15.1.

Listing 15.1. Test odbiornika podczerwieni

```
// Listing 15.1

int receiverpin = 11;          // pin nr 1 odbiornika podczerwieni połączono
                               // z pinem cyfrowym nr 11 na płytce Arduino
❶ #include <IRremote.h>        // biblioteka
❷ IRrecv irrecv(receiverpin); // tworzy obiekt funkcji IRrecv
❸ decode_results results;

void setup()
{
```

```
   Serial.begin(9600);
   irrecv.enableIRIn(); // uruchamia odbiornik podczerwieni
}

void loop()
{
❹  if (irrecv.decode(&results)) // sprawdza, czy otrzymano sygnał IR
   {
❺     Serial.print(results.value, HEX); // wyświetla kod sygnału IR w oknie
                                         // monitora portu szeregowego
      Serial.print(" ");
      irrecv.resume(); // odbiera następną wartość
   }
}
```

Powyższy szkic jest stosunkowo prosty, ponieważ większość zadań jest realizowana w tle przez zastosowaną bibliotekę. W wierszu ❹ sprawdzamy, czy odbiornik otrzymał sygnał wysłany przez nadajnik. Jeśli tak, szesnastkowa reprezentacja otrzymanej wartości jest wyświetlana w oknie monitora portu szeregowego w wierszu ❺. W wierszach ❶, ❷ i ❸ szkic aktywuje bibliotekę komunikacji na podczerwień i tworzy obiekt funkcji tej biblioteki używany w dalszej części kodu.

Testowanie układu

Po wysłaniu szkicu na płytkę Arduino należy otworzyć okno monitora portu szeregowego, skierować pilot w stronę odbiornika i nacisnąć kolejno kilka przycisków. Po każdym naciśnięciu przycisku w oknie monitora portu szeregowego powinien pojawić się odpowiedni kod. Na przykład na rysunku 15.4 pokazano wyniki wyświetlone po naciśnięciu przycisków 1, 2 i 3.

Rysunek 15.4. Dane wynikowe wyświetlone przez kod z listingu 15.1 po naciśnięciu trzech różnych przycisków pilota

W tabeli 15.1 wymieniono kody wysyłane przez prosty pilot do telewizora firmy Sony. Kody zostaną wykorzystane w kolejnych szkicach. Okazuje się jednak (widać to w danych wynikowych szkicu z listingu 15.1), że każdy kod jest

powtarzany trzykrotnie. To typowe działanie dla pilotów na podczerwień firmy Sony — każde naciśnięcie przycisku pilota powoduje wysłanie odpowiedniego kodu trzy razy. Powtórzenia można ignorować za pomocą prostej konstrukcji w kodzie szkicu, jednak nie warto teraz tracić czasu — przejdźmy do następnego projektu.

Tabela 15.1. Przykładowe kody wysyłane przez pilot firmy Sony

Przycisk	Kod	Przycisk	Kod
Włącz/wyłącz	A90	7	610
Wycisz	290	8	E10
1	10	9	110
2	810	0	910
3	410	Zwiększ głośność	490
4	C10	Zmniejsz głośność	C90
5	210	Następny kanał	90
6	A10	Poprzedni kanał	890

Projekt nr 50: zdalne sterowanie systemem Arduino na podczerwień

Ten projekt ma zademonstrować sposób sterowania cyfrowymi pinami wyjściowymi za pomocą pilota na podczerwień. Projekt umożliwi użytkownikowi sterowanie pinami cyfrowymi nr 2 – 7 za pomocą przycisków od 2 do 7 pilota firmy Sony. Po naciśnięciu przycisku na pilocie odpowiedni cyfrowy pin wyjściowy zmieni swój stan na `HIGH` na jedną sekundę, a następnie wróci do stanu `LOW`. Ten projekt będzie dobrym punktem wyjścia lub przynajmniej źródłem inspiracji przy wielu innych projektach wymagających zdalnego sterowania.

Sprzęt

Projekt wymaga takich samych urządzeń, jakich użyto w projekcie testowym na początku tego rozdziału, oraz niestandardowego modułu LED opracowanego w projekcie nr 28. (Zamiast tego modułu zawsze można użyć odrębnych diod LED, silników lub innych urządzeń wyjściowych).

Szkic

W środowisku IDE należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 50 — zdalne sterowanie systemem Arduino na podczerwień
int receiverpin = 11;          // pin nr 1 odbiornika podczerwieni połączono
                               // z pinem cyfrowym nr 11 na płytce Arduino
```

```
#include <IRremote.h>
IRrecv irrecv(receiverpin); // tworzy obiekt funkcji IRrecv
decode_results results;

void setup()
{
   irrecv.enableIRIn(); // uruchamia odbiornik
   for (int z = 2 ; z < 8 ; z++) // konfiguruje piny cyfrowe
   {
      pinMode(z, OUTPUT);
   }
}

❶  void translateIR()
// podejmuje działania zależnie od otrzymanego kodu IR
// (na podstawie kodów stosowanych w produktach Sony)
{
   switch(results.value)
   {
❷     case 0x810: pinOn(2); break; // 2
      case 0x410: pinOn(3); break; // 3
      case 0xC10: pinOn(4); break; // 4
      case 0x210: pinOn(5); break; // 5
      case 0xA10: pinOn(6); break; // 6
      case 0x610: pinOn(7); break; // 7
   }
}

❸  void pinOn(int pin) // włącza pin cyfrowy na 1 sekundę
{
   digitalWrite(pin, HIGH);
   delay(1000);
   digitalWrite(pin, LOW);
}

void loop()
{
❹  if (irrecv.decode(&results)) // sprawdza, czy otrzymano sygnał IR
   {
      translateIR();
❺  for (int z = 0 ; z < 2 ; z++) // ignoruje drugie i trzecie powtórzenie kodu
      {
         irrecv.resume(); // odbiera następną wartość
      }
   }
}
```

Szkic dla tego projektu składa się z trzech głównych części. Po pierwsze, program czeka na sygnał od pilota (w wierszu ❹). Po otrzymaniu tego sygnału szkic przekazuje uzyskaną wartość do funkcji `translateIR()` (w wierszu ❶), określającej, który przycisk został naciśnięty i jakie działania należy podjąć.

Warto zwrócić uwagę na sposób porównywania (w wierszu ❷) kodów szesnastkowych przy użyciu biblioteki. Zastosowane tutaj wartości szesnastkowe odpowiadają kodom zwróconym podczas testowania szkicu z listingu 15.1. W razie otrzymania kodów właściwych przyciskom od 2 do 7 szkic wywołuje funkcję `pinOn()` zdefiniowaną w wierszu ❸. Funkcja włącza na jedną sekundę odpowiedni pin cyfrowy.

Jak już wspomniałem, piloty firmy Sony wysyłają kod trzy razy dla każdego naciśnięcia przycisku — w związku z tym w wierszu ❺ zastosowano prostą pętlę ignorującą drugie i trzecie wystąpienie kodu. I wreszcie warto zwrócić uwagę na przedrostek `0x` poprzedzający liczby szesnastkowe w wyrażeniach `case` w wierszu ❷.

UWAGA *Liczby szesnastkowe należą do systemu o podstawie 16. Do wyrażenia liczb z tego systemu używa się cyfr od 0 do 9 oraz liter od A do F. Na przykład dziesiętna liczba 10 w systemie szesnastkowym jest reprezentowana przez literę A, dziesiętna liczba 15 jest reprezentowana przez literę F, a liczba 16 w systemie dziesiętnym ma postać 10. Liczby szesnastkowe stosowane w kodzie szkicu należy poprzedzać przedrostkiem `0x`.*

Rozszerzanie szkicu

Szkic można rozszerzyć o obsługę dodatkowych przycisków potrzebnych do sterowania robotem gąsienicowym. Wystarczy użyć kodu z listingu 15.1 do określenia kodów reprezentujących poszczególne przyciski, a następnie dodać obsługę nowych kodów w wyrażeniu `switch-case`.

Projekt nr 51: budowa zdalnie sterowanego robota

Aby zademonstrować sposób integracji systemu sterowania pilotem na podczerwień z istniejącym projektem, dodamy mechanizm zdalnego sterowania do robota opisanego w projekcie nr 40. Tym razem zamiast z góry określać kierunek i czas ruchu robota, szkic będzie decydował o działaniach urządzenia na podstawie sygnałów wysyłanych za pomocą pilota do telewizora firmy Sony.

Sprzęt

Projekt wymaga takich samych urządzeń, jakich użyto podczas budowy robota w ramach projektu nr 40, oraz dodatkowego modułu odbiornika podczerwieni (stosowanego w projektach opisanych wcześniej w tym rozdziale). W poniższym szkicu zaimplementowano rozwiązanie, dzięki któremu robot będzie reagował na naciskanie przycisków pilota według schematu: przycisk 2 powoduje ruch w przód, przycisk 8 ruch w tył, przycisk 4 obrót w lewo, a przycisk 6 obrót w prawo.

Szkic

Po ponownym połączeniu elementów robota i dodaniu odbiornika podczerwieni należy wpisać i wysłać na płytkę Arduino następujący szkic:

```
// Projekt nr 51 — budowa zdalnie sterowanego robota

int receiverpin = 11;        // pin nr 1 odbiornika podczerwieni połączono
                             // z pinem cyfrowym nr 11 na płytce Arduino
#include <IRremote.h>

IRrecv irrecv(receiverpin); // tworzy obiekt funkcji IRrecv
decode_results results;

int m1speed     = 6; // piny cyfrowe używane do sterowania szybkością
int m2speed     = 5;
int m1direction = 7; // piny cyfrowe używane do sterowania kierunkiem pracy silników
int m2direction = 4;

void setup()
{
   pinMode(m1direction, OUTPUT);
   pinMode(m2direction, OUTPUT);
   irrecv.enableIRIn(); // uruchamia odbiornik podczerwieni
}

void goForward(int duration, int pwm)
{
   digitalWrite(m1direction, HIGH); // ruch w przód
   digitalWrite(m2direction, HIGH); // ruch w przód
   analogWrite(m1speed, pwm);       // z określoną szybkością
   analogWrite(m2speed, pwm);
   delay(duration);                 // czas ruchu
   analogWrite(m1speed, 0);         // następnie robot zatrzymuje się
   analogWrite(m2speed, 0);
}

void goBackward(int duration, int pwm)
{
   digitalWrite(m1direction, LOW); // ruch w tył
   digitalWrite(m2direction, LOW); // ruch w tył
   analogWrite(m1speed, pwm);      // z określoną szybkością
   analogWrite(m2speed, pwm);
   delay(duration);
   analogWrite(m1speed, 0);        // następnie robot zatrzymuje się
   analogWrite(m2speed, 0);
}

void rotateRight(int duration, int pwm)
```

```
{
   digitalWrite(m1direction, HIGH); // ruch w przód
   digitalWrite(m2direction, LOW);  // ruch w tył
   analogWrite(m1speed, pwm);       // z określoną szybkością
   analogWrite(m2speed, pwm);
   delay(duration);                 // czas ruchu
   analogWrite(m1speed, 0);         // następnie robot zatrzymuje się
   analogWrite(m2speed, 0);
}

void rotateLeft(int duration, int pwm)
{
   digitalWrite(m1direction, LOW);  // ruch w tył
   digitalWrite(m2direction, HIGH); // ruch w przód
   analogWrite(m1speed, pwm);       // z określoną szybkością
   analogWrite(m2speed, pwm);
   delay(duration);                 // czas ruchu
   analogWrite(m1speed, 0);         // następnie robot zatrzymuje się
   analogWrite(m2speed, 0);
}

// funkcja translateIR podejmuje działania zależnie od otrzymanego kodu IR (według
// standardów Sony)
void translateIR()
{
   switch(results.value)
   {
      case 0x810: goForward(250, 255);   break; // 2
      case 0xC10: rotateLeft(250, 255);  break; // 4
      case 0xA10: rotateRight(250, 255); break; // 6
      case 0xE10: goBackward(250, 255);  break; // 8
   }
}

void loop()
{
   if (irrecv.decode(&results)) // sprawdza, czy otrzymano sygnał IR
   {
      translateIR();
      for (int z = 0 ; z < 2 ; z++) // ignoruje powtórzenia kodu
      {
         irrecv.resume(); // odbiera następną wartość
      }
   }
}
```

Szkic powinien wyglądać znajomo. W największym uproszczeniu zamiast włączać piny cyfrowe, szkic wywołuje funkcje sterujące silnikami, które były już stosowane do sterowania robotem gąsienicowym w rozdziale 12.

Co dalej?

Po zrealizowaniu projektów opisanych w tym rozdziale powinieneś doskonale rozumieć, jak wysyłać polecenia do systemu Arduino za pomocą zdalnego pilota na podczerwień. Połączenie tych umiejętności z uzyskaną do tej pory wiedzą (i wiedzą, którą nabędziesz przy okazji kolejnych projektów) pozwoli zastąpić wszystkie fizyczne formy przekazywania danych wejściowych, na przykład przyciski, zdalnymi pilotami.

Nie ma jednak powodu spoczywać na laurach. W następnym rozdziale wykorzystamy platformę Arduino do realizacji zadania, które na pierwszy rzut oka wydaje się zbyt futurystyczne i nieosiągalne dla zwykłych śmiertelników — do budowy systemów identyfikacji na podstawie częstotliwości radiowych.

16

Odczytywanie etykiet RFID

W tym rozdziale:

- dowiesz się, jak implementować czytniki etykiet RFID na bazie platformy Arduino;
- nauczysz się zapisywać zmienne w pamięci EEPROM na płytce Arduino;
- zaprojektujesz framework dla systemu dostępu do etykiet RFID na bazie Arduino.

RFID (od ang. *radio-frequency identification*) to bezprzewodowy system wykorzystujący pole elektromagnetyczne do przekazywania danych pomiędzy obiektami bez konieczności bezpośredniego kontaktu tych obiektów. Okazuje się, że na bazie platformy Arduino można zbudować urządzenie odczytujące typowe etykiety i karty RFID. Takie urządzenie może być stosowane w systemach kontroli dostępu lub sterowania cyfrowymi pinami wyjściowymi. Być może miałeś już okazję używać kart RFID na przykład w formie kart dostępu umożliwiających otwieranie drzwi lub kart transportu publicznego zbliżanych do czytników w autobusach lub tramwajach. Na rysunku 16.1 pokazano kilka przykładów etykiet i kart RFID.

Rysunek 16.1. Przykładowe urządzenia korzystające ze standardu RFID

Wewnętrzna budowa urządzeń RFID

Wewnątrz urządzenia RFID znajduje się bardzo mały układ scalony z pamięcią. Dostęp do tego układu wymaga specjalistycznego czytnika. Większość etykiet RFID nie zawiera baterii — są zasilane energią pola magnetycznego generowanego przez czytnik RFID. Pole jest generowane przez cewkę, która jednocześnie pełni funkcję anteny odbierającej dane przesyłane pomiędzy kartą a czytnikiem. Na rysunku 16.2 pokazano cewkę anteny czytnika RFID, który będzie stosowany w tym rozdziale.

Rysunek 16.2. Czytnik RFID używany w tym rozdziale

Czytnik kart, który będzie stosowany w tym rozdziale, można kupić w sklepie Seeed Studio (*http://www.seeedstudio.com/*; produkt nr ELB149C5M). Czytnik jest tani i łatwy w użyciu. Ponieważ urządzenie używa częstotliwości 125 kHz, koniecznie należy wybrać etykiety RFID operujące na tej samej częstotliwości (na przykład produkt nr RFR103B2B w sklepie Seeed Studio).

Testowanie sprzętu

W tym podrozdziale połączymy czytnik RFID z płytką Arduino, a następnie przetestujemy jego działanie za pomocą prostego szkicu odczytującego sygnał z kart RFID i wysyłającego te dane do monitora portu szeregowego.

Schemat

Na rysunku 16.3 pokazano połączenia niezbędne do budowy modułu RFID.

Rysunek 16.3. Połączenia w ramach modułu RFID

Testowanie schematu

Po połączeniu czytnika RFID z płytką Arduino możemy przetestować ten obwód, umieszczając czarną zworkę w taki sposób, aby łączyła lewy i środkowy pin w sekcji zworek. By połączyć czytnik RFID z płytką Arduino, musisz wykonać następujące kroki (do łączenia obu urządzeń trzeba użyć przewodów połączeniowych z końcówkami żeńskimi i męskimi):

1. Wtyczkę cewki umieść w gnieździe anteny.
2. Pin GND czytnika połącz z pinem GND na płytce Arduino.
3. Pin VCC czytnika połącz z pinem 5 V na płytce Arduino.
4. Pin RX połącz z pinem cyfrowym nr 0 (D0) na płytce Arduino.
5. Pin TX połącz z pinem cyfrowym nr 1 (D1) na płytce Arduino.

UWAGA *Na czas wysyłania szkiców na płytkę Arduino połączoną z czytnikiem RFID należy usunąć przewód łączący pin RX tego czytnika z pinem cyfrowym nr 0 na płytce Arduino. Przewód można ponownie podłączyć dopiero po prawidłowym wysłaniu szkicu. Usunięcie połączenia jest konieczne, ponieważ pin D0 jest używany przez system Arduino także do komunikacji z komputerem i otrzymywania szkiców.*

Szkic testowy

Po wpisaniu szkicu z listingu 16.1 należy go wysłać na płytkę Arduino.

Listing 16.1. Szkic testujący moduł RFID

```
// Listing 16.1
int data1 = 0;

void setup()
{
   Serial.begin(9600);
}

void loop()
{
   if (Serial.available() > 0) {
      data1 = Serial.read();
      // wyświetla otrzymaną liczbę
      Serial.print(" ");
      Serial.print(data1, DEC);
   }
}
```

Weryfikacja danych w oknie monitora portu szeregowego

Należy teraz otworzyć okno monitora portu szeregowego i wykonać kilka ruchów etykietą RFID nad cewką. Wyświetlony wynik powinien przypominać dane widoczne na rysunku 16.4.

Rysunek 16.4. Przykładowe dane wynikowe wygenerowane przez szkic z listingu 16.1

Warto zwrócić uwagę na wyświetlenie aż 14 liczb w oknie monitora portu szeregowego. Sekwencja liczb to unikatowy identyfikator etykiety RFID. W dalszych szkicach właśnie na podstawie tej sekwencji będziemy identyfikować etykiety. Warto teraz zapisać liczby wyświetlone dla każdej z etykiet, ponieważ będziemy potrzebowali tych identyfikatorów w kilku następnych projektach.

Projekt nr 52: budowa prostego systemu kontroli dostępu na bazie technologii RFID

Czas sprawdzić system RFID w praktycznych zastosowaniach. W tym projekcie pokażę, jak wywoływać zdarzenia systemu Arduino w momencie odczytania prawidłowej etykiety RFID. W kodzie szkicu zapisano identyfikatory dwóch etykiet RFID — w momencie odczytania jednej z tych kart przez czytnik szkic wyświetli w oknie monitora portu szeregowego komunikat *Karta zaakceptowana*. W razie przyłożenia do czytnika niewłaściwej karty w oknie monitora pojawi się komunikat *Karta odrzucona*. Rozwiązanie przygotowane w ramach tego projektu będzie punktem wyjścia dla dalszych prac przy użyciu technologii RFID (polegających między innymi na uzupełnianiu istniejących projektów o ten mechanizm).

Szkic

W środowisku IDE należy wpisać i wysłać następujący szkic. Warto jednak pamiętać o konieczności zastąpienia znaków x w obu tablicach (w wierszach ❶ i ❷) wartościami składającymi się na identyfikatory stosowanych kart RFID, które można sprawdzić za pomocą szkicu opisanego we wcześniejszej części tego rozdziału. (Same tablice zostały omówione w rozdziale 6.).

```
// Projekt nr 52 — budowa prostego systemu kontroli dostępu na bazie technologii RFID

int data1 = 0;
int ok = -1;

// identyfikatory etykiet RFID można sprawdzić za pomocą szkicu z listingu 16.1
❶ int tag1[14] = {x, x, x, x, x, x, x, x, x, x, x, x, x, x};
❷ int tag2[14] = {x, x, x, x, x, x, x, x, x, x, x, x, x, x};

int newtag[14] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // tablica używana do
                                                // porównywania odczytanego identyfikatora

void setup()
{
   Serial.flush(); // należy wyczyścić bufor portu szeregowego,
                   // w przeciwnym razie pierwszy odczyt mógłby być nieprawidłowy
```

```
      Serial.begin(9600);
   }

❸  boolean comparetag(int aa[14], int bb[14])
   {
      boolean ff = false;
      int fg = 0;
      for (int cc = 0 ; cc < 14 ; cc++)
      {
         if (aa[cc] == bb[cc])
         {
            fg++;
         }
      }
      if (fg == 14)
      {
         ff = true;
      }
      return ff;
   }

❹  void checkmytags() // porównuje oba zapisane identyfikatory z właśnie odczytanym
                      // identyfikatorem
   {
      ok = 0; // ta zmienna ułatwia podjęcie decyzji;
              // jeśli ma wartość 1, identyfikatory są takie; jeśli ma wartość 0,
              // identyfikatory są różne;
              // wartość –1 oznacza, że nie odczytano identyfikatora do sprawdzenia
      if (comparetag(newtag, tag1) == true)
      {
❺       ok++;
      }
      if (comparetag(newtag, tag2) == true)
      {
❻       ok++;
      }
   }

   void loop()
   {
      ok = -1;
      if (Serial.available() > 0) // jeśli umieszczono etykietę w sąsiedztwie czytnika,
      {
        // należy odczytać liczbę przekazaną za pośrednictwem pinu szeregowego RX
         delay(100); // czas potrzebny na przesłanie danych
                     // z bufora portu szeregowego
         for (int z = 0 ; z < 14 ; z++) // odczytuje pozostałe liczby identyfikatora
❼        {
            data1 = Serial.read();
            newtag[z] = data1;
```

```
      }
      Serial.flush(); // usuwa z bufora ewentualne kolejne odczyty
                      // czas porównać identyfikatory
❽   checkmytags();
    }
❾   // tutaj należy podjąć kroki zależnie od dopasowania identyfikatorów
    if (ok > 0) // jeśli identyfikatory są takie same
    {
      Serial.println("Karta zaakceptowana");
      ok = -1;
    }
    else if (ok == 0) // jeśli identyfikatory są różne
    {
      Serial.println("Karta odrzucona");
      ok = -1;
    }
}
```

Działanie szkicu

Po umieszczeniu etykiety RFID w sąsiedztwie czytnika liczby składające się na identyfikator tej etykiety są wysyłane za pośrednictwem portu szeregowego. W wierszu ❼ szkic pobiera wszystkie 14 liczb i umieszcza je w tablicy `newtag[]`. Identyfikator jest następnie porównywany z dwoma identyfikatorami zapisanymi w wierszach ❶ i ❷. Do porównania identyfikatorów wykorzystano funkcję `checkmytags()` (zdefiniowaną w wierszu ❹ i wywoływaną w wierszu ❽). Każda para identyfikatorów jest porównywana za pomocą funkcji `comparetag()` zdefiniowanej w wierszu ❸.

Funkcja `comparetag()` otrzymuje na wejściu dwie tablice liczb i zwraca wartość logiczną określającą, czy te tablice są takie same (`true`), czy różne (`false`). Jeśli odczytany identyfikator pasuje do któregoś z identyfikatorów zapisanych w kodzie, zmiennej `ok` jest przypisywana wartość 1 w wierszach ❺ i ❻. I wreszcie w wierszu ❾ szkic decyduje o dalszym działaniu w zależności od ewentualnego dopasowania identyfikatorów.

Po wysłaniu szkicu i ponownym połączeniu pinu cyfrowego nr 0 na płytce Arduino z pinem RX czytnika RFID (patrz rysunek 16.3) należy otworzyć okno monitora portu szeregowego i umieścić kartę RFID w sąsiedztwie czytnika. Wynik wyświetlony na ekranie powinien przypominać dane widoczne na rysunku 16.5.

Rysunek 16.5. Dane wynikowe wygenerowane przez szkic dla projektu nr 52

Zapisywanie danych we wbudowanej pamięci EEPROM systemu Arduino

Zmienne definiowane i stosowane w szkicach dla platformy Arduino przechowują dane tylko do momentu zresetowania tego systemu lub wyłączenia zasilania. Co w takim razie można zrobić, aby zachować te wartości na przyszłość? Jak trwale zapisać na przykład stały kod PIN, który użytkownik będzie mógł wpisywać za pośrednictwem klawiatury numerycznej (jak w rozdziale 9.)? W takim przypadku warto skorzystać z *pamięci EEPROM* (od ang. *electrically erasable programmable read-only memory*). Pamięć EEPROM przechowuje wartości zmiennych w mikrokontrolerze ATmega328. Wartości zapisane w tej pamięci są zachowywane także po wyłączeniu zasilania.

Pamięć EEPROM na płytce Arduino może przechowywać wartości 1024 zmiennych jednobajtowych na pozycjach oznaczonych numerami od 0 do 1023. Wystarczy przypomnieć sobie, że jeden bajt może reprezentować liczbę całkowitą z przedziału od 0 do 255, aby zrozumieć, że opisywana pamięć wprost doskonale nadaje się do przechowywania liczb składających się na identyfikator etykiety RFID. Warunkiem korzystania z pamięci EEPROM w szkicach jest wywołanie biblioteki *EEPROM* (dołączonej do środowiska Arduino IDE) za pomocą następującego wyrażenia:

```
#include <EEPROM.h>
```

Zapisanie wartości w pamięci EEPROM wymaga już tylko jednego prostego wyrażenia:

```
EEPROM.write(a, b);
```

Parametr `a` reprezentuje pozycję (z przedziału od 0 do 1023), na której dana wartość ma zostać zapisana, natomiast parametr `b` to zmienna zawierająca bajt danych, który ma być zapisany w pamięci EEPROM na wskazanej pozycji.

Aby uzyskać dane z pamięci EEPROM, należy użyć wyrażenia w postaci:

```
value = EEPROM.read(position);
```

Powyższe wyrażenie odczytuje dane zapisane w pamięci EEPROM na pozycji `position` i zapisuje je w zmiennej `value`.

UWAGA *Żywotność pamięci EEPROM jest dość ograniczona, zatem pamięć po pewnej liczbie operacji zapisu przestanie działać! Według deklaracji producenta (firmy Atmel) pamięć EEPROM platformy Arduino obsługuje maksymalnie 100 tys. cykli zapisu i odczytu na każdej pozycji. Operacje odczytu nie wpływają na czas działania tej pamięci.*

Odczytywanie i zapisywanie danych w pamięci EEPROM

W tym punkcie przeanalizujemy przykład szkicu zapisującego dane w pamięci EEPROM i odczytującego te dane. Należy teraz wpisać i wysłać szkic z listingu 16.2.

Listing 16.2. Szkic demonstrujący działanie pamięci EEPROM

```
// Listing 16.2
#include <EEPROM.h>
int zz;

void setup()
{
   Serial.begin(9600);
   randomSeed(analogRead(0));
}

void loop()
{
   Serial.println("Zapisywanie liczb losowych...");
   for (int i = 0; i < 1024; i++)
   {
      zz = random(255);
❶     EEPROM.write(i, zz);
   }
   Serial.println();
   for (int a = 0; a < 1024; a++)
   {
❷     zz = EEPROM.read(a);
      Serial.print("Pozycja w pamięci EEPROM: ");
      Serial.print(a);
      Serial.print(" zawiera wartość ");
❸     Serial.println(zz);
      delay(25);
   }
}
```

W pętli w wierszu ❶ liczba losowa z przedziału od 0 do 255 jest zapisywana na każdej pozycji w pamięci EEPROM. Zapisane wartości są następnie odczytywane w drugiej pętli w wierszu ❷ oraz wyświetlane w oknie monitora portu szeregowego (w wierszu ❸).

Po wysłaniu tego szkicu na platformę Arduino należy otworzyć okno monitora portu szeregowego, w którym powinny zostać wyświetlone dane podobne do tych z rysunku 16.6.

Możemy teraz przystąpić do realizacji projektu korzystającego z pamięci EEPROM.

Rysunek 16.6. Przykładowe dane wynikowe wygenerowane przez szkic z listingu 16.2

Projekt nr 53: budowa systemu kontroli dostępu RFID z pamięcią „ostatniej akcji”

Mimo że w projekcie nr 52 pokazałem, jak używać etykiet RFID do sterowania urządzeniem, na przykład światłem lub zamkiem elektrycznym, musimy pamiętać o tym, że ponowne uruchomienie lub wyłączenie systemu spowoduje utratę zapisanych danych. Jeśli na przykład po włączeniu światła zostało wyłączone zasilanie, po ponownym włączeniu zasilania światło nie zostanie automatycznie włączone. Tym razem chcemy, aby po ponownym włączeniu zasilania system Arduino pamiętał i przywrócił swój stan sprzed wyłączenia zasilania. Spróbujmy rozwiązać ten problem w tym projekcie.

Szkic tego projektu będzie zapisywał ostatnie działanie (akcję) w pamięci EEPROM (tą akcją może być na przykład zamknięcie lub otwarcie zamka). Po ponownym uruchomieniu szkicu po awarii zasilania lub zresetowaniu platformy Arduino system automatycznie odtworzy stan zapisany w pamięci EEPROM.

Szkic

W środowisku Arduino IDE należy wpisać i wysłać następujący szkic. Także tym razem trzeba zastąpić znaki x w obu tablicach w wierszach ❶ i ❷ liczbami składowymi identyfikatorów obu etykiet RFID (tak jak w projekcie nr 52).

```
// Projekt nr 53 — budowa systemu kontroli dostępu RFID z pamięcią „ostatniej akcji”

#include <EEPROM.h>
int data1 = 0;
int ok = -1;
int lockStatus = 0;

// identyfikatory etykiet RFID można sprawdzić za pomocą szkicu z listingu 16.1
❶ int tag1[14] = {x, x, x, x, x, x, x, x, x, x, x, x, x, x};
❷ int tag2[14] = {x, x, x, x, x, x, x, x, x, x, x, x, x, x};
```

```
int newtag[14] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // tablica używana do
                                                // porównywania odczytanego identyfikatora

void setup()
{
   Serial.flush();
   Serial.begin(9600);
   pinMode(13, OUTPUT);
❸  checkLock();
}

// funkcja comparetag() porównuje dwie tablice i zwraca wartość true, jeśli są identyczne
// funkcja sprawdza się podczas porównywania identyfikatorów RFID
boolean comparetag(int aa[14], int bb[14])
{
   boolean ff = false;
   int fg = 0;
   for (int cc = 0; cc < 14; cc++)
   {
      if (aa[cc] == bb[cc])
      {
         fg++;
      }
   }
   if (fg == 14)
   {
      ff = true;
   }
   return ff;
}

void checkmytags()
// porównuje oba zapisane identyfikatory z właśnie odczytanym identyfikatorem
{
   ok = 0;
   if (comparetag(newtag, tag1) == true)
   {
      ok++;
   }
   if (comparetag(newtag, tag2) == true)
   {
      ok++;
   }
}

❹ void checkLock()
{
   Serial.print("Stan systemu po ponownym uruchomieniu ");
   lockStatus = EEPROM.read(0);
   if (lockStatus == 1)
```

```
    {
       Serial.println("- zablokowany");
       digitalWrite(13, HIGH);
    }
    if (lockStatus == 0)
    {
       Serial.println("- odblokowany");
       digitalWrite(13, LOW);
    }
    if ((lockStatus != 1) && (lockStatus != 0))
    {
       Serial.println("Błąd pamięci EEPROM - zmień płytkę Arduino");
    }
 }

 void loop()
 {
    ok = -1;
    if (Serial.available() > 0) // jeśli umieszczono etykietę w sąsiedztwie czytnika,
    {
       // należy odczytać liczbę przekazaną za pośrednictwem pinu szeregowego RX
       delay(100);
       for (int z = 0; z < 14; z++) // odczytuje pozostałe liczby identyfikatora
       {
          data1 = Serial.read();
          newtag[z] = data1;
       }
       Serial.flush(); // zapobiega wielokrotnym odczytom tego samego identyfikatora
       // czas porównać identyfikatory
       checkmytags();
    }
❺   if (ok > 0) // jeśli identyfikatory są takie same
    {
       lockStatus = EEPROM.read(0);
       if (lockStatus == 1) // jeśli zablokowany, odblokowuje
❻      {
          Serial.println("Stan - odblokowany");
          digitalWrite(13, LOW);
          EEPROM.write(0, 0);
       }
       if (lockStatus == 0)
❼      {
          Serial.println("Stan - zablokowany");
          digitalWrite(13, HIGH);
          EEPROM.write(0, 1);
       }
       if ((lockStatus != 1) && (lockStatus != 0))
❽      {
          Serial.println("Błąd pamięci EEPROM - zmień płytkę Arduino");
       }
```

```
    }
    else if (ok == 0) // jeśli identyfikatory są różne
    {
       Serial.println("Nieprawidłowa etykieta");
       ok = -1;
    }
    delay(500);
}
```

Działanie szkicu

Powyższy szkic jest zmodyfikowaną wersją programu z projektu nr 52. Do symulacji stanu systemu wykorzystano wbudowaną diodę LED, która dobrze demonstruje urządzenie włączane i wyłączane po każdym odczytaniu prawidłowego identyfikatora RFID. Po odczytaniu i dopasowaniu identyfikatora status zamka jest zmieniany w wierszu ❺. Stan zamka jest zapisywany na pierwszej pozycji w pamięci EEPROM. Stan jest reprezentowany przez wartość liczbową: 0 oznacza odblokowany zamek; 1 oznacza zablokowany zamek. Stan zamka zmienia się (z zablokowanego na odblokowany i odwrotnie) po każdym odczycie prawidłowego identyfikatora RFID w wierszach ❻ i ❼.

Szkic zawiera też kod zabezpieczający system na wypadek, gdyby pamięć EEPROM przestała działać. Jeśli wartość zwrócona po odczytaniu zawartości pamięci EEPROM jest różna od 0 i 1, szkic wyświetla odpowiedni komunikat w wierszu ❽. Co więcej, stan systemu jest sprawdzany po ponownym uruchomieniu szkicu w funkcji `checkLock()` w wierszach ❶, ❷, ❸ i ❹. Funkcja odczytuje wartość zapisaną w pamięci EEPROM, określa na tej podstawie ostatni stan systemu (zamka) i według tego ustawia bieżący stan (zablokowany lub odblokowany).

Co dalej?

W tym rozdziale jeszcze raz wykorzystaliśmy platformę Arduino do budowy prostego urządzenia, które bez tego systemu wymagałoby bardzo złożonego projektu. Wiedza przekazana w tym miejscu wystarczy do stosowania identyfikacji kart RFID w wielu innych projektach i pozwoli tworzyć profesjonalne systemy kontroli dostępu i sterowania cyfrowymi pinami wyjściowymi. Zastosowania tej technologii ponownie zademonstruję w rozdziale 18.

17

Magistrale danych

W tym rozdziale:

- dowiesz się, jak działa magistrala I^2C;
- nauczysz się używać pamięci EEPROM (od ang. *electrically erasable read-only memory*) i ekspandera portów podczas korzystania z magistrali I^2C;
- dowiesz się, jak działa magistrala SPI;
- nauczysz się stosowania cyfrowego rezystora nastawnego (reostatu) z magistralą SPI.

Platforma Arduino komunikuje się z pozostałymi urządzeniami za pośrednictwem *magistrali danych*, czyli systemu połączeń umożliwiających wymianę danych pomiędzy co najmniej dwoma urządzeniami. Magistrala danych umożliwia łączenie platformy Arduino z różnymi czujnikami, urządzeniami wejścia-wyjścia i innymi komponentami.

Dla systemu Arduino najważniejsze są dwie magistrale: *SPI* (od ang. *serial peripheral interface*) oraz **I^2C** (od ang. *inter-integrated circuit*). Wiele popularnych czujników i urządzeń zewnętrznych komunikuje się z platformą Arduino właśnie za pośrednictwem tych magistral.

Magistrala I²C

Magistrala I²C, znana też jako magistrala *TWI* (od ang. *two wire interface*) to proste, łatwe w użyciu urządzenie wykorzystywane do przesyłania danych. Dane są przesyłane pomiędzy urządzeniami i platformą Arduino za pośrednictwem dwóch przewodów: *SDA* (linia danych) i *SCL* (linia zegara). W przypadku płytki Arduino Uno linia SDA jest obsługiwana przez pin A4, a linia SCL przez pin A5 (patrz rysunek 17.1). Niektóre nowsze płytki z serii R3 dodatkowo oferują dedykowane piny magistrali I²C w lewym górnym rogu (taka lokalizacja zapewnia łatwy dostęp do magistrali).

Rysunek 17.1. Piny magistrali I²C na płytce Arduino Uno

W kontekście magistrali I²C system Arduino jest traktowany jako *urządzenie nadrzędne*, natomiast układy scalone połączone z tą magistralą są traktowane jako *urządzenia podrzędne*. Każde urządzenie podrzędne ma przypisany adres w formie liczby szesnastkowej, dzięki czemu system Arduino może niezależnie komunikować się z poszczególnymi urządzeniami. W większości przypadków każde urządzenie oferuje pewien przedział adresów magistrali I²C, spośród których można wybrać jeden stosowany adres (informacje na temat dostępnych adresów można znaleźć w dołączonej instrukcji). Wybór adresu polega na odpowiednim ustawieniu zworek łączących piny układu scalonego.

UWAGA *Ponieważ system Arduino jest zasilany prądem o napięciu 5 V, także urządzenia połączone z magistralą I²C muszą być zasilane takim prądem lub przynajmniej tolerować to napięcie. Warto to sprawdzić, kontaktując się ze sprzedawcą lub producentem urządzenia, które ma być połączone z magistralą I²C.*

Warunkiem korzystania z magistrali I²C jest zastosowanie biblioteki `Wire` (dołączonej do środowiska Arduino IDE):

```
#include <Wire.h>
```

Magistralę można następnie aktywować w funkcji `void setup()`:

```
Wire.begin();
```

Dane są wysyłane za pośrednictwem magistrali po jednym bajcie. Aby wysłać bajt danych z platformy Arduino do urządzenia połączonego z magistralą, należy użyć trzech funkcji:

1. Pierwsza funkcja inicjuje komunikację. Na wejściu tej funkcji należy przekazać adres (w formie liczby szesnastkowej) urządzenia podrzędnego połączonego z magistralą szeregową, na przykład `0x50`:

```
Wire.beginTransmission(adres);
```

2. Druga funkcja wysyła 1 bajt danych z systemu Arduino do urządzenia zaadresowanego za pomocą poprzedniej funkcji (parametr *dane* to zmienna zawierająca bajt danych; istnieje oczywiście możliwość wysłania większej ilości danych, ale każdy bajt wymaga osobnego wywołania funkcji `Wire.write()`):

```
Wire.write(dane);
```

3. I wreszcie po zakończeniu wysyłania danych do określonego urządzenia należy zakończyć transmisję za pomocą funkcji:

```
Wire.endTransmission();
```

Aby zażądać danych wysłanych przez urządzenie połączone z magistralą I^2C do systemu Arduino, należy najpierw rozpocząć transmisję za pomocą wspomnianej już funkcji `Wire.beginTransmission(adres)`, a następnie użyć następującej funkcji (gdzie *x* to liczba żądanych bajtów danych):

```
Wire.requestFrom(adres, x);
```

Zapisanie otrzymanego bajta danych w zmiennej wymaga użycia następującego wyrażenia:

```
incoming = Wire.read(); // zmienna incoming będzie zawierała otrzymany bajt danych
```

Transakcję należy teraz zakończyć za pomocą funkcji `Wire.endTransmission()`. Wszystkie wymienione funkcje zostaną użyte w najbliższym projekcie.

Projekt nr 54: stosowanie zewnętrznej pamięci EEPROM

W rozdziale 16. wykorzystaliśmy wewnętrzną pamięć EEPROM platformy Arduino do zapobiegania utracie danych wskutek awarii zasilania lub ponownego uruchomienia tego systemu. Wewnętrzna pamięć EEPROM systemu Arduino umożliwia przechowywanie tylko 1024 bajtów danych. Aby zapisać więcej danych, wystarczy zastosować zewnętrzną pamięć EEPROM — właśnie takie rozwiązanie wykorzystamy w tym projekcie.

W roli zewnętrznej pamięci EEPROM użyjemy układu 24LC512 EEPROM firmy Microchip Technology, który oferuje możliwość przechowywania 64 kB (65 536 bajtów) danych (patrz rysunek 17.2). Pamięć jest dostępna w takich sklepach jak Digi-Key (numer produktu: 24LC512-I/P-ND) czy element14 (numer produktu: 1660008).

Rysunek 17.2. Pamięć 24LC512 EEPROM firmy Microchip Technology

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- pamięć 24LC512 EEPROM firmy Microchip Technology;
- płytka uniwersalna;
- dwa rezystory 4,7 kΩ;
- kondensator ceramiczny 100 nF;
- różne przewody;
- płytka Arduino i przewód USB.

Schemat

Budowa tego obwodu wymaga umieszczenia rezystora 4,7 kΩ pomiędzy pinami 5 V i SCL oraz pomiędzy pinami 5 V i SDA (patrz rysunek 17.3).

Rysunek 17.3. Schemat obwodu dla projektu nr 54

Adres układu scalonego pamięci 24LC512 EEPROM w ramach magistrali po części zależy od sposobu połączenia tego układu z obwodem. Trzy ostatnie bity adresu magistrali są określane na podstawie stanu pinów A2, A1 i A0. Jeśli wymienione piny zostaną połączone z pinem GND, ich stan będzie równy 0; jeśli zostaną połączone z pinem 5 V, ich stan będzie równy 1.

Pierwsze cztery bity adresu są stałe i wynoszą `1010`. W naszym obwodzie adres magistrali będzie reprezentowany przez liczbę binarną `1010000`, czyli odpowiednik liczby szesnastkowej `0x50`. Oznacza to, że możemy stosować wartość `0x50` w roli adresu magistrali w kodzie szkicu.

Szkic

Mimo że w zastosowanej w tym projekcie zewnętrznej pamięci EEPROM można przechowywać nawet 64 kB danych, nasz szkic ma na celu zademonstrowanie zaledwie wycinka możliwości tego układu, zatem będzie zapisywał i odczytywał dane na pierwszych 20 pozycjach tej pamięci.

W środowisku Arduino IDE należy wpisać i wysłać następujący szkic:

```
// Projekt nr 54 — stosowanie zewnętrznej pamięci EEPROM
❶ #include <Wire.h>
#define chip1 0x50
```

```
unsigned int pointer;
byte d=0;

void setup()
{
❷   Serial.begin(9600);
    Wire.begin();
}

void writeData(int device, unsigned int address, byte data)
// zapisuje bajt danych reprezentowany przez zmienną data w pamięci EEPROM
// podłączonej do magistrali I2C (adres device);
// bajt jest zapisywany na pozycji address
{
❸   Wire.beginTransmission(device);
    Wire.write((byte)(address >> 8));    // lewa część wskaźnika address
    Wire.write((byte)(address & 0xFF)); // prawa część wskaźnika
    Wire.write(data);
    Wire.endTransmission();
    delay(10);
}

❹   byte readData(int device, unsigned int address)
// odczytuje bajt danych z pozycji address w pamięci
// podłączonej do magistrali I2C i reprezentowanej przez adres device
{
    byte result; // zwrócona wartość
    Wire.beginTransmission(device);
    Wire.write((byte)(address >> 8));    // lewa część wskaźnika address
    Wire.write((byte)(address & 0xFF)); // prawa część wskaźnika
    Wire.endTransmission();
❺   Wire.requestFrom(device,1);
    result = Wire.read();
    return result; // zwraca odczytaną wartość jako wynik funkcji readData
}

void loop()
{
    Serial.println("Zapisywanie danych...");
    for (int a=0; a<20; a++)
    {
        writeData(chip1,a,a);
    }
    Serial.println("Odczytywanie danych...");
    for (int a=0; a<20; a++)
    {
        Serial.print("Pozycja w pamięci EEPROM ");
        Serial.print(a);
        Serial.print(" zawiera ");
        d=readData(chip1,a);
```

```
      Serial.println(d, DEC);
   }
}
```

Przeanalizujmy teraz działanie tego szkicu. W wierszu ❶ aktywujemy bibliotekę i definiujemy adres magistrali I²C dla układu scalonego pamięci EEPROM — adres jest reprezentowany przez stałą `chip1`. W wierszu ❷ uruchamiamy kolejno monitor portu szeregowego i magistralę I²C. Funkcje niestandardowe `writeData()` i `readData()` znacznie ułatwiają implementację tego szkicu i jednocześnie stanowią źródło kodu wielokrotnego użytku, który będzie można wykorzystać w przyszłych projektach wymagających zewnętrznej pamięci EEPROM. Funkcje odpowiadają odpowiednio za zapisywanie danych w pamięci EEPROM i odczytywanie danych z tej pamięci.

Funkcja `writeData()` w wierszu ❸ rozpoczyna komunikację z układem EEPROM, wysyła adres docelowy dla bajta danych w pamięci EEPROM za pomocą dwóch wywołań funkcji `Wire.write()`, wysyła bajt danych do zapisania i wreszcie kończy transmisję danych.

Funkcja `readData()` w wierszu ❹ operuje na magistrali I²C w taki sam sposób jak funkcja `writeData()`, tyle że zamiast wysyłać bajt danych do pamięci EEPROM, używa funkcji `Wire.requestFrom()` do odczytania danych (w wierszu ❺). I wreszcie bajt danych odczytany z pamięci EEPROM jest przypisywany zmiennej `result`, która zaraz potem jest zwracana przez funkcję `readData()`.

Wynik

W ciele funkcji `void loop()` szkic wykonuje 20 iteracji pętli, z których każda zapisuje jedną wartość w pamięci EEPROM. Bezpośrednio potem szkic wykonuje kolejną pętlę, która tym razem odpowiada za odczytywanie i wyświetlanie wartości w oknie monitora portu szeregowego (patrz rysunek 17.4).

Rysunek 17.4. Wynik działania szkicu z projektu nr 54

Projekt nr 55: stosowanie układu ekspandera portów

Ekspander portów to kolejny przydatny układ scalony sterowany za pośrednictwem magistrali I²C. Układy tego typu są projektowane z myślą o zwiększaniu liczby cyfrowych pinów wyjściowych. W tym projekcie zostanie użyty 16-bitowy ekspander portów MCP23017 firmy Microchip Technology (patrz rysunek 17.5), który umożliwia zwiększenie liczby wyjść cyfrowych platformy Arduino o 16 pinów. Komponent jest dostępny w takich sklepach jak Newark (numer produktu: 31K2959) czy element14 (numer produktu: 1332088).

Rysunek 17.5. Ekspander portów MCP23017 firmy Microchip Technology

W tym projekcie połączymy układ MCP23017 z płytką Arduino, aby zademonstrować sposób sterowania 16 pinami wyjściowymi ekspandera portów za pośrednictwem systemu Arduino. Każde wyjście ekspandera portów można traktować jak standardowy wyjściowy pin cyfrowy na płytce Arduino.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- płytka Arduino i przewód USB;
- płytka uniwersalna;
- różne przewody;
- ekspander portów MCP23017 firmy Microchip Technology;
- dwa rezystory 4,7 kΩ;
- (opcjonalnie) pewna liczba rezystorów 560 Ω i tyle samo diod LED.

Schemat

Podstawowy schemat obwodu obejmującego układ MCP23017 pokazano na rysunku 17.6. Tak jak w przypadku zewnętrznej pamięci EEPROM użytej w projekcie nr 54, adres magistrali I²C można określić, stosując odpowiednie połączenia. Tym razem piny od 15. do 17. układu MCP23017 zostaną połączone z pinem GND, zatem adres tego komponentu będzie miał wartość `0x20`.

Rysunek 17.6. Podstawowy schemat obwodu dla projektu nr 55

Podczas łączenia układu MCP23017 warto korzystać z diagramu układu pinów przygotowanego przez producenta (patrz rysunek 17.7). Łatwo zauważyć, że 16 pinów wejściowych podzielono na dwie grupy: od GPA7 do GPA0 po prawej stronie i od GPB0 do GPB7 po lewej stronie. Do części lub wszystkich pinów wyjściowych można podłączyć diody LED (za pośrednictwem rezystorów 560 Ω), aby wizualnie zademonstrować aktywowanie poszczególnych wyjść.

Rysunek 17.7. Diagram układu pinów ekspandera portów MCP23017

Szkic

W środowisku Arduino IDE należy wpisać i wysłać następujący szkic:

```
// Projekt nr 55 — stosowanie układu ekspandera portów

#include "Wire.h"
#define mcp23017 0x20

void setup()
{
   Wire.begin(); // aktywuje magistralę I2C
   // konfiguruje układ MCP23017
   // ustawia piny jako wyjściowe
   Wire.beginTransmission(mcp23017);
   Wire.write(0x00); // rejestr IODIRA
   Wire.write(0x00); // ustawia wszystkie piny z grupy A jako wyjściowe
   Wire.write(0x00); // ustawia wszystkie piny z grupy B jako wyjściowe
❷  Wire.endTransmission();
}

void loop()
{
   Wire.beginTransmission(mcp23017);
   Wire.write(0x12);
❸  Wire.write(255); // grupa A
❹  Wire.write(255); // grupa B
   Wire.endTransmission();
   delay(1000);

   Wire.beginTransmission(mcp23017);
   Wire.write(0x12);
   Wire.write(0); // grupa A
   Wire.write(0); // grupa B
   Wire.endTransmission();
   delay(1000);
}
```

Warunkiem korzystania z ekspandera portów MCP23017 jest konfiguracja tego układu w wierszach od ❶ do ❷ w ciele funkcji `void setup()`. Aby włączać i wyłączać piny wyjściowe w obu grupach, wystarczy wysłać bajt reprezentujący stan poszczególnych pinów; pierwszy bajt określa stan pinów z grupy od GPA0 do GPA7, drugi bajt określa stan pinów z grupy od GPB0 do GPB7.

Podczas ustawiania stanu poszczególnych pinów każdą grupę należy traktować jako liczbę binarną (patrz podrozdział „Krótkie wprowadzenie w świat liczb binarnych” w rozdziale 6.). Oznacza to, że włączenie pinów od 7. do 4. wymaga wysłania liczby binarnej 11110000 (czyli liczby dziesiętnej 240) za pośrednictwem funkcji `Wire.write()` (takie rozwiązanie zastosowano w wierszu ❸ dla grupy pinów od GPA0 do GPA7 oraz w wierszu ❹ dla grupy pinów od GPB0 do GPB7).

Magistrala I²C jest używana do komunikacji przez setki urządzeń. Skoro znamy już podstawy stosowania tej magistrali, możemy korzystać z tej formy komunikacji podczas budowy urządzeń na bazie platformy Arduino.

Magistrala SPI

Magistrala SPI różni się od magistrali I²C tym, że może być używana do jednoczesnego wysyłania do urządzenia i odbierania z niego danych z różnymi szybkościami (zależnymi od stosowanego mikrokontrolera). Okazuje się jednak, że także w przypadku magistrali SPI istnieje *podział na urządzenie nadrzędne i podrzędne* — system Arduino pełni funkcję *urządzenia nadrzędnego* i określa, z którym *urządzeniem podrzędnym* będzie się komunikował.

Łączenie pinów

Każde urządzenie podłączone do magistrali SPI używa czterech pinów do komunikacji z urządzeniem nadrzędnym: *MOSI* (od ang. *master-out, slave-in*), *MISO* (od ang. *master-in, slave-out*), *SCK* (zegar) i *SS* lub *CS* (odpowiednio od ang. *slave select* lub *chip select*). Piny magistrali SPI są połączone z płytką Arduino zgodnie ze schematem pokazanym na rysunku 17.8.

Rysunek 17.8. Piny magistrali SPI na płytce Arduino Uno

Na rysunku 17.9 pokazano typowe połączenie pojedynczego urządzenia z platformą Arduino za pośrednictwem magistrali SPI. Dla magistrali SPI zarezerwowano piny od D11 do D13 na płytce Arduino. W roli czwartego pinu można wykorzystać dowolny inny pin cyfrowy (zwykle stosuje się pin D10, ponieważ znajduje się obok pinów zarezerwowanych dla magistrali SPI).

Rysunek 17.9. Typowe połączenie urządzenia z platformą Arduino za pośrednictwem magistrali SPI

UWAGA *Podobnie jak w przypadku urządzeń połączonych z magistralą I^2C, urządzenia połączone z magistralą SPI muszą albo być zasilane prądem o napięciu 5 V, albo przynajmniej być przystosowane do działania przy takim napięciu (ponieważ płytka Arduino jest zasilana prądem o napięciu 5 V). Warto więc sprawdzić napięcie zasilania urządzeń łączonych za pośrednictwem tej magistrali u sprzedawcy lub producenta.*

Implementacja obsługi magistrali SPI

Przeanalizujmy teraz sposób implementacji obsługi magistrali SPI w ramach szkicu. Zanim będzie możliwe korzystanie z tej formy komunikacji, należy wywołać kilka funkcji. Musimy najpierw dołączyć bibliotekę magistrali SPI (instalowaną wraz ze środowiskiem Arduino IDE):

```
#include "SPI.h"
```

Należy następnie wskazać pin używany jako pin SS i ustawić tryb działania tego pinu jako wyjście (w ciele funkcji `void setup`). Ponieważ w opisywanym przykładzie będzie stosowane tylko jedno urządzenie SPI, użyjemy pinu D10 i ustawimy jego początkowy stan na `HIGH` — takie rozwiązanie jest niezbędne z uwagi na aktywowanie większości urządzeń SPI niskim stanem pinu SS:

```
pinMode(10, OUTPUT);
digitalWrite(10, HIGH);
```

Poniższe wywołanie funkcji aktywuje magistralę SPI:

```
SPI.begin();
```

I wreszcie musimy określić w szkicu sposób wysyłania i odbierania danych. Niektóre urządzenia SPI wymagają wysyłania danych z najbardziej znaczącym bitem (ang. *most significant bit — MSB*) na pierwszej pozycji; inne urządzenia wymagają przesyłania danych z najbardziej znaczącym bitem na końcu. (Kwestia najbardziej znaczącego bitu została wyjaśniona w podrozdziale „Krótkie wprowadzenie w świat liczb binarnych" w rozdziale 6.). Oznacza to, że w funkcji `void setup` należy użyć następującego wywołania (po wywołaniu funkcji `SPI.begin`):

```
SPI.setBitOrder(order);
```

Za pośrednictwem parametru *`order`* należy przekazać wartość `MSBFIRST` lub `MSBLAST`.

Wysyłanie danych do urządzenia SPI

Aby wysłać dane do urządzenia SPI, należy najpierw ustawić stan `LOW` pinu SS — w ten sposób informujemy urządzenie SPI, że urządzenie nadrzędne (w tym przypadku płytka Arduino) nawiąże komunikację. Możemy następnie wysłać bajty danych do tego urządzenia za pomocą następującego wyrażenia — to samo wywołanie należy zastosować osobno dla każdego wywoływanego bajta:

```
SPI.transfer(byte);
```

Po zakończeniu komunikacji z urządzeniem należy ponownie ustawić stan `HIGH` pinu SS, aby zasygnalizować urządzeniu podrzędnemu koniec komunikacji z płytką Arduino.

Każde urządzenie podłączone do magistrali SPI wymaga osobnego pinu SS. Jeśli na przykład obwód obejmuje dwa urządzenia SPI, drugie z tych urządzeń może wykorzystywać w roli pinu SS na przykład pin D9 — schemat takiego połączenia pokazano na rysunku 17.10.

Rysunek 17.10. Dwa urządzenia SPI połączone z jedną płytką Arduino

Podczas wymiany danych z drugim urządzeniem podrzędnym należy zmienić stan pinu D9 (zamiast pinu D10) używanego w roli pinu SS przed rozpoczęciem komunikacji i po jej zakończeniu.

Projekt nr 56 demonstruje sposób stosowania magistrali SPI łącznie z cyfrowym rezystorem nastawnym (reostatem).

Projekt nr 56: stosowanie cyfrowego rezystora nastawnego

Rezystor nastawny pod wieloma względami przypomina potencjometry opisane w rozdziale 4. Najważniejsza różnica polega na tym, że rezystor nastawny udostępnia dwa piny: wejściowy i wyjściowy. W tym projekcie użyjemy cyfrowego rezystora nastawnego do ustawiania oporu z poziomu szkicu (zamiast ręcznie regulować gałkę potencjometru). Rezystory nastawne często są stosowane w układach sterowania głośnością w tych systemach audio, w których głośność jest regulowana za pomocą przycisków (nie pokręteł). Tolerancja rezystora nastawnego jest dużo większa niż w przypadku standardowych rezystorów o stałej oporności — w niektórych przypadkach tolerancja jest wyższa nawet o około 20 procent.

W tym projekcie zostanie użyty rezystor MCP4162 firmy Microchip Technology (patrz rysunek 17.11). Układ MCP4162 jest oferowany w wersjach z różnymi wartościami oporności; w tym przykładzie posłużymy się wersją 10 kΩ. Produkt jest dostępny w wielu sklepach internetowych, w tym w sklepie Newark (numer produktu: 77M2766) oraz element14 (numer produktu: 1840698). Wartość rezystancji można zmieniać w 255 krokach; każdy krok reprezentuje opór równy w przybliżeniu 40 Ω. Aby ustawić określoną rezystancję, należy wysłać 2 bajty danych: bajt rozkazu (z wartością 0) oraz bajt wartości (z wartością z przedziału od 0 do 255). Rezystor MCP4162 dysponuje pamięcią nieulotną, zatem po odłączeniu i ponownym podłączeniu zasilania będzie stosowana ostatnio ustawiona wartość.

Rysunek 17.11. Rezystor nastawny MCP4162 firmy Microchip Technology

Za pomocą rezystora nastawnego będziemy sterowali jasnością diody LED.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- płytka Arduino i przewód USB;
- płytka uniwersalna;
- różne przewody;
- rezystor nastawny MCP4162 firmy Microchip Technology;
- rezystor 560 Ω;
- dioda LED.

Schemat

Schemat obwodu dla tego projektu pokazano na rysunku 17.12. Piny układu MCP4162 są numerowane, począwszy od lewego górnego rogu. Pin nr 1 jest oznaczony kropką nadrukowaną na lewo od logo firmy Microchip (patrz rysunek 17.11).

Rysunek 17.12. Schemat obwodu dla projektu nr 56

Szkic

W środowisku Arduino IDE należy wpisać i wysłać następujący szkic:

```
// Projekt nr 56 — stosowanie cyfrowego rezystora nastawnego

❶ #include "SPI.h" // niezbędna biblioteka
int ss=10; // pin nr 10 będzie pełnił funkcję pinu wyboru urządzenia podrzędnego
           // magistrali SPI
```

```
int del=200; // opóźnienie pomiędzy kolejnymi krokami skali jasności diody LED

void setup()
{
❷   SPI.begin();
    pinMode(ss, OUTPUT); // pin SS
    digitalWrite(ss, HIGH); // pin SS jest aktywowany stanem LOW, zatem
                            // początkowo ustawiamy stan HIGH
❸   SPI.setBitOrder(MSBFIRST);
    // układ MCP4162 wymaga wysyłania najbardziej znaczącego bitu (MSB) jako pierwszego
}

❹   void setValue(int value)
{
    digitalWrite(ss, LOW);
    SPI.transfer(0); // wysyła bajt rozkazu
    SPI.transfer(value); // wysyła bajt wartości (od 0 do 255)
    digitalWrite(ss, HIGH);
}

void loop()
{
❺   for (int a=0; a<256; a++)
    {
      setValue(a);
      delay(del);
    }
❻   for (int a=255; a>=0; a--)
    {
      setValue(a);
      delay(del);
    }
}
```

Przeanalizujmy teraz działanie tego kodu. Szkic rozpoczyna się od kodu konfigurującego magistralę SPI w wierszach ❶ i ❷. W wierszu ❸ ustawiono model kodowania bajtów zgodny z wymaganiami układu MPC4162. Aby ułatwić ustawianie oporności, w wierszu ❹ zdefiniowano niestandardową funkcję, która otrzymuje na wejściu krok rezystancji (od 0 do 255) i przekazuje tę wartość do rezystora nastawnego MCP4162. I wreszcie szkic zawiera dwie pętle zmieniające opór rezystora nastawnego od zera do wartości maksymalnej (w wierszu ❺) i ponownie do zera (w wierszu ❻). Ostatni fragment kodu powinien powodować zwiększanie i zmniejszanie jasności diody LED — dioda powinna płynnie zwiększać jasność do maksymalnej i gasnąć przez cały czas działania szkicu.

Co dalej?

W tym rozdziale wyjaśniłem i zademonstrowałem dwie ważne metody komunikacji platformy Arduino z urządzeniami zewnętrznymi. Po lekturze tego rozdziału możesz łączyć swoją płytkę Arduino z najróżniejszymi zaawansowanymi czujnikami i innymi komponentami dostępnymi na rynku i używającymi właśnie opisanych magistral. Jednym z najbardziej popularnych komponentów jest układ scalony zegara czasu rzeczywistego, który umożliwia urządzeniom precyzyjne śledzenie czasu — stosowanie tych układów będzie tematem rozdziału 18. Do dzieła!

18

Zegary czasu rzeczywistego

W tym rozdziale:

- nauczysz się ustawiać i odczytywać datę i godzinę z modułu zegara czasu rzeczywistego;
- odkryjesz nowe sposoby łączenia urządzeń z platformą Arduino;
- zbudujesz własny zegar cyfrowy;
- zbudujesz system rejestracji czasu pracy przy użyciu technologii RFID.

Układ scalony *zegara czasu rzeczywistego* (ang. *real-time clock — RTC*) to niewielkie urządzenie mierzące czas i stwarzające mnóstwo ciekawych możliwości dla projektantów urządzeń na bazie systemu Arduino. Po ustawieniu bieżącej daty i godziny moduł RTC zapewnia (na żądanie) dostęp do precyzyjnych danych.

Na rynku można znaleźć wiele różnych układów scalonych zegarów czasu rzeczywistego, które różnią się przede wszystkim precyzją. W tym rozdziale będziemy stosowali układ Maxim DS3232, który nie wymaga żadnych zewnętrznych obwodów poza zapasową baterią. Wspomniany zegar jest wyjątkowo precyzyjny

i niezawodny. Układ DS3232 jest dostępny w formie gotowej płytki w wielu sklepach detalicznych. Na rysunku 18.1 pokazano wersję przygotowaną przez firmę Freetronics (*http://www.freetronics.com/rtc/*).

Rysunek 18.1. Moduł układu scalonego zegara czasu rzeczywistego

Łączenie modułu RTC

Łączenie modułu RTC z płytką Arduino jest bardzo proste, ponieważ oba komponenty komunikują się za pośrednictwem magistrali I^2C (omówionej w rozdziale 17.). Wystarczą zaledwie cztery przewody — piny GND i VCC należy połączyć z pinami GND i 5 V na płytce Arduino, a piny SDA i SCL należy połączyć z pinami analogowymi A4 i A5. Projekt modułu zegara czasu rzeczywistego eliminuje konieczność stosowania dodatkowych rezystorów podciągających, chyba że moduł jest łączony z płytką Arduino za pomocą długich przewodów. W takim przypadku należy połączyć (przy użyciu lutownicy) końcówki oznaczone etykietą *Pullups: SDA and SCL* na spodzie płytki modułu, aby włączyć wbudowane rezystory podciągające magistrali I^2C.

Dla wygody warto rozważyć montaż modułu zegara czasu rzeczywistego na pustej płytce ProtoShield, tak aby można było łatwo integrować ten komponent z pozostałymi urządzeniami wykorzystywanymi w projekcie. Koniecznie należy zadbać także o zapasową baterię — w przeciwnym razie godzina zostanie utracona w momencie wyłączenia zasilania obwodu!

Projekt nr 57: wyświetlanie daty i godziny na podstawie zegara czasu rzeczywistego

W tym projekcie pokażę, jak ustawić datę i godzinę w układzie zegara czasu rzeczywistego oraz jak uzyskiwać i wyświetlać te dane w oknie monitora portu szeregowego. Informacje o dacie i godzinie mogą być bardzo przydatne w wielu różnych projektach, na przykład w rejestratorach temperatur i budzikach.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- płytka Arduino i przewód USB;
- różne przewody;
- moduł zegara czasu rzeczywistego Maxim DS3232.

Szkic

Moduł zegara RTC należy połączyć zgodnie ze schematem opisanym we wcześniejszej części tego rozdziału. Musimy następnie wpisać następujący szkic, ale *nie powinniśmy jeszcze wysyłać go na płytkę Arduino*:

```
// Projekt nr 57 — wyświetlanie daty i godziny na podstawie zegara czasu rzeczywistego
❶ #include "Wire.h"
#define DS3232_I2C_ADDRESS 0x68

// konwertuje liczby dziesiętne na zakodowane wartości binarne
❷ byte decToBcd(byte val)
{
   return( (val/10*16) + (val%10) );
}

// konwertuje zakodowane wartości binarne na liczby dziesiętne
byte bcdToDec(byte val)
{
   return( (val/16*10) + (val%16) );
}

void setup()
{
   Wire.begin();
   Serial.begin(9600);

   // ustawia początkową godzinę:
   // DS3232 sekundy, minuty, godziny, dzień tygodnia, dzień miesiąca, miesiąc, rok
❸  setDS3232time(0, 56, 23, 3, 30, 10, 12);
}

❹ void setDS3232time(byte second, byte minute, byte hour, byte dayOfWeek, byte
dayOfMonth, byte month, byte year)
{
   // ustawia datę i godzinę w module DS3232
   Wire.beginTransmission(DS3232_I2C_ADDRESS);
   Wire.write(0); // określa, że następna wartość ma trafić do rejestru sekund
   Wire.write(decToBcd(second));      // ustawia sekundy
   Wire.write(decToBcd(minute));      // ustawia minuty
   Wire.write(decToBcd(hour));        // ustawia godziny
```

```
   Wire.write(decToBcd(dayOfWeek));  // ustawia dzień tygodnia (1 = niedziela,
                                     // 7 = sobota)
   Wire.write(decToBcd(dayOfMonth)); // ustawia dzień miesiąca (od 1 do 31)
   Wire.write(decToBcd(month));      // ustawia miesiąc
   Wire.write(decToBcd(year));       // ustawia rok (od 0 do 99)
   Wire.endTransmission();
}

❺ void readDS3232time(byte *second,
byte *minute,
byte *hour,
byte *dayOfWeek,
byte *dayOfMonth,
byte *month,
byte *year)
{
   Wire.beginTransmission(DS3232_I2C_ADDRESS);
   Wire.write(0); // ustawia wskaźnik rejestru DS3232 na pozycji 00h
   Wire.endTransmission();
   Wire.requestFrom(DS3232_I2C_ADDRESS, 7);
   // żąda siedmiu bajtów danych od modułu DS3232, począwszy od rejestru 00h
   *second     = bcdToDec(Wire.read() & 0x7f);
   *minute     = bcdToDec(Wire.read());
   *hour       = bcdToDec(Wire.read() & 0x3f);
   *dayOfWeek  = bcdToDec(Wire.read());
   *dayOfMonth = bcdToDec(Wire.read());
   *month      = bcdToDec(Wire.read());
   *year       = bcdToDec(Wire.read());
}

void displayTime()
{
   byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
   // pobiera dane z modułu DS3232
❻  readDS3232time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month,
   &year);

   // wysyła uzyskane dane do monitora portu szeregowego
   Serial.print(hour, DEC);
   // podczas wyświetlania konwertuje wartość zmiennej typu byte na liczbę dziesiętną
   Serial.print(":");
   if (minute<10)
   {
      Serial.print("0");
   }
   Serial.print(minute, DEC);
   Serial.print(":");
   if (second<10)
   {
      Serial.print("0");
```

```
    }
    Serial.print(second, DEC);
    Serial.print(" ");
    Serial.print(dayOfMonth, DEC);
    Serial.print("/");
    Serial.print(month, DEC);
    Serial.print("/");
    Serial.print(year, DEC);
    Serial.print(" Dzień tygodnia: ");
    switch(dayOfWeek){
       case 1:
          Serial.println("niedziela");
          break;
       case 2:
          Serial.println("poniedziałek");
          break;
       case 3:
          Serial.println("wtorek");
          break;
       case 4:
          Serial.println("środa");
          break;
       case 5:
          Serial.println("czwartek");
          break;
       case 6:
          Serial.println("piątek");
          break;
       case 7:
          Serial.println("sobota");
          break;
    }
}

void loop()
{
    displayTime();  // wyświetla w oknie monitora portu szeregowego datę i godzinę
                    // pobrane z zegara czasu rzeczywistego
    delay(1000);    // cykl jest powtarzany co sekundę
}
```

Działanie szkicu

Na pierwszy rzut oka szkic może sprawiać wrażenie dość skomplikowanego, jednak w rzeczywistości wcale nie jest trudny do zrozumienia. W wierszu ❶ importujemy bibliotekę I^2C i ustawiamy wartość `0x68` jako adres modułu RTC w ramach magistrali I^2C. W wierszu ❷ dwie funkcje niestandardowe konwertują liczby dziesiętne na wartości zakodowane w systemie binarnym (ang. *binarycoded decimal — BCD*) i zwracają te wartości. Konwersja jest konieczna, ponieważ moduł DS3232 przechowuje wartości właśnie w formacie BCD.

W wierszu ❸ zastosowano funkcję `setDS3232time` do przekazania informacji o dacie i godzinie do układu scalonego zegara RTC:

```
setDS3232time(second, minute, hour, dayOfWeek, dayOfMonth, month, year)
```

Aby użyć tej funkcji, wystarczy umieścić niezbędne dane w różnych parametrach. Parametr *dayOfWeek* musi zawierać liczbę z przedziału od 1 do 7 reprezentującą dni tygodnia odpowiednio od niedzieli do soboty. Parametr reprezentujący rok powinien zawierać liczbę dwucyfrową — w tym przykładzie zastosowano wartość 13 reprezentującą rok 2013. (Wartością domyślną tego parametru jest 20). Na wejściu tej funkcji można przekazywać albo stałe wartości (jak w powyższym szkicu), albo zmienne typu `byte` z odpowiednimi parametrami.

Oznacza to, że aby ustawić godzinę w module RTC, należy przekazać wartości reprezentujące bieżącą datę i godzinę na wejściu funkcji `setDS3232time` w wierszu ❹. Szkic można teraz wysłać na płytkę Arduino. Po wysłaniu szkicu należy poprzedzić wywołanie funkcji `setDS3232time` (w wierszu ❹) znakami //, czyli umieścić to wywołanie w komentarzu, i ponownie wysłać szkic na platformę Arduino. Dzięki temu zapisane w szkicu data i godzina nie będą ponownie ustawiane po każdym uruchomieniu systemu Arduino!

I wreszcie funkcja `readDS3232time` w wierszu ❺ odczytuje datę i godzinę z modułu RTC i umieszcza poszczególne wartości składowe w zmiennych typu `byte`. Uzyskane w ten sposób wartości są następnie używane w ciele funkcji `displayTime` (w wierszu ❻), która wyświetla zawartość zmiennych w oknie monitora portu szeregowego.

Należy teraz wysłać szkic na płytkę Arduino i otworzyć okno monitora portu szeregowego. Wyświetlony wynik powinien przypominać dane widoczne na rysunku 18.2.

Rysunek 18.2. Dane wynikowe wygenerowane przez projekt nr 57

Kod szkicu przygotowanego dla projektu nr 57 może stanowić dobry punkt wyjścia dla innych projektów wymagających zegara czasu rzeczywistego. Funkcje `decToBcd`, `bcdToDec`, `readDS3232time` i `setDS3232time` można kopiować do projektów

realizowanych w przyszłości. To jedna z zalet platformy Arduino — wystarczy raz opracować jakąś przydatną funkcję, aby następnie wielokrotnie używać jej w innych projektach (najczęściej po wprowadzeniu drobnych zmian lub nawet bez konieczności stosowania jakichkolwiek modyfikacji).

MODUŁ PROTOSCREWSHIELD

Z czasem budowane projekty (coraz bardziej skomplikowane) będą się składały z coraz większej liczby modułów dodatkowych dla platformy Arduino i urządzeń zewnętrznych. Łączenie tych komponentów będzie wymagało trudnej do opanowania plątaniny przewodów. Jednym z najlepszych sposobów radzenia sobie z tym problemem jest zastosowanie modułu ProtoScrewShield firmy Wingshield Industries (*http://wingshieldindustries.com/*) dla platformy Arduino — patrz rysunek 18.3.

Rysunek 18.3. Moduł ProtoScrewShield na płytce Arduino

Komponent składa się z dwóch części, po jednej dla każdego wiersza gniazd na płytce Arduino. Po umieszczeniu tego komponentu na płytce Arduino nadal można stosować dodatkowe moduły. Dzięki modułowi ProtoScrewShield do pinów wejścia-wyjścia na płytce Arduino można łatwo (za pomocą złączy śrubowych) podłączać dodatkowe urządzenia zewnętrzne, jak czujniki czy silniki.

Projekt nr 58: tworzenie prostego zegara cyfrowego

W tym projekcie użyjemy funkcji z projektu nr 57 do wyświetlania daty i godziny na standardowym znakowym module LCD (podobnym do tego, który wykorzystano w projekcie nr 44 podczas budowy odbiornika GPS).

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- płytka Arduino i przewód USB;
- różne przewody;
- płytka uniwersalna;
- ProtoScrewShield lub podobny produkt;
- moduł LCD lub odpowiedni moduł dodatkowy firmy Freetronics;
- moduł zegara czasu rzeczywistego (opisany wcześniej w tym rozdziale).

Musimy najpierw odtworzyć obwód z projektu nr 57. Jeśli do tej pory moduł RTC był łączony z płytką Arduino za pomocą przewodów, należy użyć modułu ProtoScrewShield do połączenia obu komponentów. Następnie trzeba umieścić moduł LCD ponad pozostałymi modułami.

Szkic

W środowisku Arduino IDE należy wpisać następujący szkic, ale nie należy go wysyłać na płytkę Arduino:

```
// Projekt nr 58 — tworzenie prostego zegara cyfrowego

#include "Wire.h"
#define DS3232_I2C_ADDRESS 0x68

❶ #include <LiquidCrystal.h>
LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );

// konwertuje liczby dziesiętne na zakodowane wartości binarne
byte decToBcd(byte val)
{
   return( (val/10*16) + (val%10) );
}

// konwertuje zakodowane wartości binarne na liczby dziesiętne
byte bcdToDec(byte val)
{
   return( (val/16*10) + (val%16) );
}

void setup()
{
   Wire.begin();
❷  lcd.begin(16, 2);
   // ustawia początkową godzinę:
   // DS3232 sekundy, minuty, godziny, dzień tygodnia, dzień miesiąca, miesiąc, rok
❸  // setDS3232time(0, 27, 0, 5, 15, 11, 12);
```

```
}

void setDS3232time(byte second, byte minute, byte hour, byte
➥dayOfWeek, byte
dayOfMonth, byte month, byte year)
{
   // ustawia datę i godzinę w module DS3232
   Wire.beginTransmission(DS3232_I2C_ADDRESS);
   Wire.write(0); // określa, że następna wartość ma trafić do rejestru sekund
   Wire.write(decToBcd(second));       // ustawia sekundy
   Wire.write(decToBcd(minute));       // ustawia minuty
   Wire.write(decToBcd(hour));         // ustawia godziny
   Wire.write(decToBcd(dayOfWeek));    // ustawia dzień tygodnia (1 = niedziela,
                                       // 7 = sobota)
   Wire.write(decToBcd(dayOfMonth)); // ustawia dzień miesiąca (od 1 do 31)
   Wire.write(decToBcd(month));        // ustawia miesiąc
   Wire.write(decToBcd(year));         // ustawia rok (od 0 do 99)
   Wire.endTransmission();
}

void readDS3232time(byte *second,
byte *minute,
byte *hour,
byte *dayOfWeek,
byte *dayOfMonth,
byte *month,
byte *year)
{
   Wire.beginTransmission(DS3232_I2C_ADDRESS);
   Wire.write(0); // ustawia wskaźnik rejestru DS3232 na pozycji 00h
   Wire.endTransmission();
   Wire.requestFrom(DS3232_I2C_ADDRESS, 7);
   // żąda siedmiu bajtów danych od modułu DS3232, począwszy od rejestru 00h
   *second     = bcdToDec(Wire.read() & 0x7f);
   *minute     = bcdToDec(Wire.read());
   *hour       = bcdToDec(Wire.read() & 0x3f);
   *dayOfWeek  = bcdToDec(Wire.read());
   *dayOfMonth = bcdToDec(Wire.read());
   *month      = bcdToDec(Wire.read());
   *year       = bcdToDec(Wire.read());
}

void displayTime()
{
   byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;

   // pobiera dane z modułu DS3232
   readDS3232time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month,
   &year);
```

```
    // wysyła dane do modułu LCD
    lcd.clear();
    lcd.setCursor(4,0);
    lcd.print(hour, DEC);
    lcd.print(":");
    if (minute<10)
    {
       lcd.print("0");
    }
    lcd.print(minute, DEC);
    lcd.print(":");
    if (second<10)
    {
       lcd.print("0");
    }
    lcd.print(second, DEC);

    lcd.setCursor(0,1);
    switch(dayOfWeek){
       case 1:
          lcd.print("Nie");
          break;
       case 2:
          lcd.print("Pon");
          break;
       case 3:
          lcd.print("Wto");
          break;
       case 4:
          lcd.print("Sro");
          break;
       case 5:
          lcd.print("Czw");
          break;
       case 6:
          lcd.print("Pia");
          break;
       case 7:
          lcd.print("Sob");
          break;
    }
    lcd.print(" ");
    lcd.print(dayOfMonth, DEC);
    lcd.print("/");
    lcd.print(month, DEC);
    lcd.print("/");
    lcd.print(year, DEC);
}

void loop()
```

```
{
   displayTime(); // wyświetla na LCD datę i godzinę pobrane z zegara czasu
                  // rzeczywistego
   delay(1000);   // cykl jest powtarzany co sekundę
}
```

Działanie szkicu i generowane wyniki

Działanie tego szkicu przypomina funkcjonowanie kodu dla projektu nr 57 z tą różnicą, że tym razem funkcja `displayTime` została przystosowana do wyświetlania daty i godziny na ekranie LCD (wcześniej wysyłała te dane do monitora portu szeregowego), a na początku szkicu (w wierszach ❶ i ❷) umieszczono wiersze potrzebne do skonfigurowania modułu LCD. (Aby przypomnieć sobie metody wyświetlania danych na module LCD, warto wrócić do rozdziału 7.). Musimy jeszcze pamiętać o konieczności dwukrotnego wysłania szkicu na płytkę Arduino — za pierwszym razem należy wysłać szkic z datą i godziną wpisanymi w wierszu ❸, a następnie szkic w wersji zawierającej ten wiersz w komentarzu. Po wysłaniu szkicu wyświetlony wynik powinien przypominać dane pokazane na rysunku 18.4.

Rysunek 18.4. Dane wynikowe wygenerowane przez projekt nr 58

Po zrealizowaniu projektów nr 57 i 58 nie powinieneś mieć najmniejszych problemów z odczytywaniem i zapisywaniem danych przechowywanych w układzie zegara czasu rzeczywistego. Spróbujmy wykorzystać zdobyte do tej pory umiejętności do zbudowania naprawdę przydatnego urządzenia.

Projekt nr 59: budowa systemu kontroli czasu pracy na bazie technologii RFID

W ramach tego projektu zbudujemy system mierzenia czasu pracy. Podczas realizacji tego projektu pokażę, jak poszczególne moduły dodatkowe dla płytki Arduino mogą ze sobą współpracować i jak komponent ProtoScrewShield ułatwia łączenie elementów elektronicznych, które nie zostały zamontowane na module. System może być używany na przykład przez dwie osoby dysponujące kartą lub etykietą RFID zbliżaną do czytnika w momencie wchodzenia do określonego pomieszczenia lub budynku (na przykład miejsca pracy lub domu) oraz opusz-

czania tego miejsca. Godziny i szczegółowe informacje o kartach można rejestrować na karcie microSD z myślą o późniejszych analizach.

Metody rejestrowania danych na kartach microSD zostały omówione w rozdziale 13., technologię RFID wprowadziłem w rozdziale 16., a zegary RTC we wcześniejszej części tego rozdziału. W tym projekcie połączymy wszystkie te urządzenia i technologie.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- płytka Arduino i przewód USB;
- różne przewody;
- moduł zegara czasu rzeczywistego (opisany wcześniej w tym rozdziale);
- moduł LCD lub odpowiedni moduł dodatkowy firmy Freetronics;
- moduł kart pamięci microSD i karta (z rozdziału 13.);
- ProtoScrewShield lub podobny produkt;
- moduł czytnika RFID i dwie etykiety (z rozdziału 16.).

Aby zbudować ten system, należy umieścić na samym dole płytkę Arduino Uno, zainstalować na niej komponent ProtoScrewShield, na nim zamontować moduł kart pamięci microSD i wreszcie na samej górze moduł LCD. Czytnik RFID należy podłączyć zgodnie z opisem zawartym w rozdziale 16. Sposób łączenia modułu RTC opisano we wcześniejszej części tego rozdziału. Gotowe urządzenie powinno wyglądać podobnie do obwodu pokazanego na rysunku 18.5.

Rysunek 18.5. Obwód systemu kontroli czasu

Szkic

W środowisku IDE należy teraz wpisać i wysłać następujący szkic. Musimy pamiętać, że na czas wysyłania szkiców na płytkę Arduino połączoną z czytnikiem RFID trzeba usunąć przewód łączący pin RX tego czytnika z pinem D0 na płytce Arduino. Przewód należy ponownie podłączyć po zakończeniu wysyłania szkicu.

```
// Projekt nr 59 — budowa systemu kontroli czasu pracy na bazie technologii RFID

❶ #include "Wire.h" // dla modułu RTC
  #define DS3232_I2C_ADDRESS 0x68
❷ #include "SD.h" // dla kart SD

  #include <LiquidCrystal.h>
  LiquidCrystal lcd( 8, 9, 4, 5, 6, 7 );
  int data1 = 0;

❸ // identyfikatory etykiet RFID można sprawdzić za pomocą szkicu z listingu 16.1
  int Mary[14] = {
     2, 52, 48, 48, 48, 56, 54, 67, 54, 54, 66, 54, 66, 3};
  int John[14] = {
     2, 52, 48, 48, 48, 56, 54, 66, 49, 52, 70, 51, 56, 3};
  int newtag[14] = {
     0,0,0,0,0,0,0,0,0,0,0,0,0,0}; // tablica używana do porównywania
                                   // odczytanego identyfikatora

  // konwertuje liczby dziesiętne na zakodowane wartości binarne
  byte decToBcd(byte val)
  {
     return( (val/10*16) + (val%10) );
  }

  // konwertuje zakodowane wartości binarne na liczby dziesiętne
  byte bcdToDec(byte val)
  {
     return( (val/16*10) + (val%16) );
  }

  void setDS3232time(byte second, byte minute, byte hour, byte dayOfWeek,
  byte
  dayOfMonth, byte month, byte year)
  {
     // ustawia datę i godzinę w module DS3232
     Wire.beginTransmission(DS3232_I2C_ADDRESS);
     Wire.write(0); // określa, że następna wartość ma trafić do rejestru sekund
     Wire.write(decToBcd(second));      // ustawia sekundy
     Wire.write(decToBcd(minute));      // ustawia minuty
     Wire.write(decToBcd(hour));        // ustawia godziny
```

```
   Wire.write(decToBcd(dayOfWeek));  // ustawia dzień tygodnia (1 = niedziela,
                                     // 7 = sobota)
   Wire.write(decToBcd(dayOfMonth)); // ustawia dzień miesiąca (od 1 do 31)
   Wire.write(decToBcd(month));      // ustawia miesiąc
   Wire.write(decToBcd(year));       // ustawia rok (od 0 do 99)
   Wire.endTransmission();
}

void readDS3232time(byte *second,
byte *minute,
byte *hour,
byte *dayOfWeek,
byte *dayOfMonth,
byte *month,
byte *year)
{
   Wire.beginTransmission(DS3232_I2C_ADDRESS);
   Wire.write(0); // ustawia wskaźnik rejestru DS3232 na pozycji 00h
   Wire.endTransmission();
   Wire.requestFrom(DS3232_I2C_ADDRESS, 7);

   // żąda siedmiu bajtów danych od modułu DS3232, począwszy od rejestru 00h
   *second     = bcdToDec(Wire.read() & 0x7f);
   *minute     = bcdToDec(Wire.read());
   *hour       = bcdToDec(Wire.read() & 0x3f);
   *dayOfWeek  = bcdToDec(Wire.read());
   *dayOfMonth = bcdToDec(Wire.read());
   *month      = bcdToDec(Wire.read());
   *year       = bcdToDec(Wire.read());
}

void setup()
{
   Serial.flush(); // należy wyczyścić bufor portu szeregowego
   Serial.begin(9600);
   Wire.begin();
   lcd.begin(16, 2);
   // ustawia początkową godzinę:
   // DS3232 sekundy, minuty, godziny, dzień tygodnia, dzień miesiąca, miesiąc, rok
   //setDS3232time(0, 27, 0, 5, 15, 11, 12);

   // sprawdza, czy karta microSD jest dostępna i gotowa do użycia
❹  if (!SD.begin(8))
   {
      lcd.print("Karta microSD BŁĄD");
      // zatrzymuje wykonywanie szkicu
      return;
   }
   lcd.print("Karta microSD OK");
   delay(1000);
```

```
   lcd.clear();
}

// porównuje dwie tablice i zwraca wartość true, jeśli są identyczne
// funkcja sprawdza się podczas porównywania identyfikatorów RFID
boolean comparetag(int aa[14], int bb[14])
{
   boolean ff=false;
   int fg=0;
   for (int cc=0; cc<14; cc++)
   {
      if (aa[cc]==bb[cc])
      {
         fg++;
      }
   }
   if (fg==14)
   {
      ff=true; // takie same wszystkie 14 elementów tablicy
   }
   return ff;
}

void wipeNewTag()
{
   for (int i=0; i<=14; i++)
   {
      newtag[i]=0;
   }
}

void loop()
{
   byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;

   if (Serial.available() > 0) // jeśli umieszczono etykietę w sąsiedztwie czytnika,
   {
      // należy odczytać liczbę przekazaną za pośrednictwem pinu szeregowego RX
      delay(100); // czas potrzebny na przesłanie danych z bufora portu szeregowego
      for (int z=0; z<14; z++) // odczytuje pozostałe liczby identyfikatora
      {
         data1=Serial.read();
         newtag[z]=data1;
      }
      Serial.flush(); // usuwa z bufora ewentualne kolejne odczyty
      // pobiera dane z modułu DS3232
      readDS3232time(&second, &minute, &hour, &dayOfWeek,
      ↪&dayOfMonth, &month,
      &year);
   }
```

```
   // tutaj należy podjąć kroki zależnie od dopasowania identyfikatorów
❺  if (comparetag(newtag, Mary) == true)
   {
      lcd.print("Witaj, Mario ");
      File dataFile = SD.open("DATA.TXT", FILE_WRITE);
      if (dataFile)
      {
         dataFile.print("Maria ");
         dataFile.print(hour);
         dataFile.print(":");
         if (minute<10) { dataFile.print("0"); }
         dataFile.print(minute);
         dataFile.print(":");
         if (second<10) { dataFile.print("0"); }
         dataFile.print(second);
         dataFile.print(" ");
         dataFile.print(dayOfMonth);
         dataFile.print("/");
         dataFile.print(month);
         dataFile.print("/");
         dataFile.print(year);
         dataFile.println();
         dataFile.close();
      }
      delay(1000);
      lcd.clear();
      wipeNewTag();
   }

   if (comparetag(newtag, John)==true)
   {
      lcd.print("Witaj, Janie ");
      File dataFile = SD.open("DATA.TXT", FILE_WRITE);
      if (dataFile)
      {
         dataFile.print("Jan   ");
         dataFile.print(hour);
         dataFile.print(":");
         if (minute<10) { dataFile.print("0"); }
         dataFile.print(minute);
         dataFile.print(":");
         if (second<10) { dataFile.print("0"); }
         dataFile.print(second);
         dataFile.print(" ");
         dataFile.print(dayOfMonth);
         dataFile.print("/");
         dataFile.print(month);
         dataFile.print("/");
         dataFile.print(year);
         dataFile.println();
         dataFile.close();
```

```
        }
        delay(1000);
        lcd.clear();
        wipeNewTag();
    }
}
```

Działanie szkicu

Po uruchomieniu tego szkicu system czeka na dane o karcie RFID przekazane przez czytnik. W przypadku rozpoznania tej karty imię jej właściciela oraz bieżące data i godzina są zapisywane w pliku przechowywanym na karcie microSD.

W wierszu ❶ dołączono bibliotekę niezbędną do obsługi magistrali I^2C i modułu zegara czasu rzeczywistego, natomiast w wierszu ❷ dołączono bibliotekę potrzebną do komunikacji z modułem kart pamięci microSD. W wierszu ❹ szkic sprawdza stan karty microSD i wyświetla odpowiedni komunikat. W wierszu ❺ odczytany identyfikator karty RFID jest porównywany z identyfikatorami obu kart zapisanymi w kodzie szkicu (w tym przypadku karty należą do Marii i Jana). Jeśli numer karty pasuje do któregoś z tych identyfikatorów, odpowiednia informacja jest zapisywana na karcie pamięci microSD. Wystarczy kilka prostych zmian, aby dodać do systemu kolejne karty — należy dopisać identyfikatory tych kart pod istniejącymi numerami w wierszu ❸, a następnie dodać odpowiednie wywołania funkcji porównującej numery (tak jak w wierszu ❺).

Aby przeanalizować zarejestrowane dane, należy skopiować plik *data.txt* z karty microSD, a następnie otworzyć go w edytorze tekstu lub zaimportować do arkusza kalkulacyjnego. Dane są sformatowane w taki sposób, że ich interpretacja nie powinna stanowić żadnego problemu (patrz rysunek 18.6).

Rysunek 18.6. Przykładowe dane wygenerowane przez projekt nr 59

Co dalej?

W tym rozdziale pokazałem, jak pracować z precyzyjnymi informacjami o dacie i godzinie dostarczanymi przez układ zegara czasu rzeczywistego (RTC). System na bazie technologii RFID, który opisałem w projekcie nr 59, może stanowić

punkt wyjścia dla budowanych przez Ciebie systemów kontroli dostępu lub nawet systemów śledzenia czasu powrotu dzieci do domu. W ostatnich dwóch rozdziałach zrealizujemy projekty, w których system Arduino będzie się komunikował ze światem za pośrednictwem internetu i sieci telefonii komórkowej.

19

Internet

W tym rozdziale:

- zbudujesz serwer WWW wyświetlający dane na stronie internetowej;
- dowiesz się, jak użyć płytki Arduino do publikowania wpisów na Twitterze;
- nauczysz się zdalnie sterować cyfrowymi pinami wyjściowymi na płytce Arduino z poziomu przeglądarki.

W tym rozdziale pokażę, jak łączyć platformę Arduino ze światem zewnętrznym za pośrednictwem internetu. W ten sposób można nie tylko wysyłać dane z platformy Arduino, ale też zdalnie sterować tym systemem z poziomu przeglądarki internetowej.

Czego potrzebujemy?

Do budowy projektów przystosowanych do komunikacji za pośrednictwem internetu będziemy potrzebowali sprzętu, przewodu i pewnych informacji.

Zacznijmy od sprzętu. Będziemy potrzebowali modułu sieciowego z układem kontrolera W5100. Możemy wybrać jedno z dwóch rozwiązań: na rynku są dostępne zarówno uniwersalne moduły sieciowe dla platformy Arduino (patrz

rysunek 19.1), jak i zintegrowane kontrolery sieciowe zgodne z płytką Arduino Uno, na przykład Freetronics EtherTen (patrz rysunek 19.2). Wybór drugiego urządzenia jest lepszym rozwiązaniem w przypadku nowych projektów lub projektów, w których duże znaczenie ma oszczędność przestrzeni i pieniędzy. Jak widać, moduł EtherTen udostępnia złącza dla modułów dodatkowych platformy Arduino, port USB, gniazdo Ethernet oraz gniazdo kart pamięci microSD.

Rysunek 19.1. Moduł sieciowy dla platformy Arduino

Rysunek 19.2. Moduł Freetronics EtherTen

Niezależnie od wybranego modułu sieciowego musimy zadbać także o standardowy przewód sieciowy 10/100 CAT5, CAT5E lub CAT6, którym połączymy moduł sieciowy z routerem lub modemem.

Będziemy też potrzebowali adresu IP bramy sieciowej routera lub modemu. W większości sieci ten adres ma następującą postać: 192.168.0.1. Inną ważną informacją jest adres IP komputera (w takim samym formacie jak adres IP routera).

I wreszcie jeśli system Arduino ma się komunikować ze światem zewnętrznym z naszego domu lub jakiejś lokalnej sieci komputerowej, będziemy potrzebowali statycznego, publicznego adresu IP dla samej płytki. Statyczny adres IP to stały adres przypisany fizycznemu połączeniu internetowemu przez operatora tego połączenia (ISP). Nie każde połączenie internetowe ma domyślnie przypisywany statyczny adres IP, zatem w razie konieczności należy skontaktować się w tej sprawie z operatorem połączenia. Jeśli operator w ogóle nie oferuje statycznych adresów IP lub jeśli zakup takiego adresu jest zbyt drogi, warto rozważyć skorzystanie z usług oferowanych przez niezależne firmy, na przykład no-ip (*http://www.noip.com/*) lub Dyn (*http://dyn.com/dns/*). Wymienione firmy oferują możliwość przypisania do serwera stałego adresu strony internetowej, który nie będzie zmieniany niezależnie od bieżącego adresu IP.

Czas przetestować wszystkie niezbędne urządzenia przy okazji realizacji projektu.

Projekt nr 60: budowa zdalnej stacji monitoringu

W projektach realizowanych we wcześniejszych rozdziałach dane były pobierane z zewnętrznych czujników między innymi w celu mierzenia temperatury i natężenia światła. W tym projekcie pokażę, jak wyświetlać te wartości na prostej stronie internetowej, którą można otworzyć na niemal wszystkich urządzeniach obsługujących przeglądarki internetowe. Projekt będzie wyświetlał wartości odczytywane za pośrednictwem analogowych pinów wejściowych oraz stan dziesięciu cyfrowych pinów wejściowych na prostej stronie internetowej. Rozwiązania opracowane w ramach tego projektu mogą stanowić dobry punkt wyjścia dla innych systemów zdalnego monitorowania.

Szkic stosowany w tym projekcie może być wykorzystany w roli frameworku dla innych rozwiązań obejmujących dodatkowe czujniki (podłączone do pinów analogowych i cyfrowych), w tym czujników temperatury, natężenia światła i przełączników. Stan tych czujników będzie wyświetlany na stronie internetowej.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- przewód USB;
- przewód sieciowy;
- płytka Arduino Uno i moduł sieciowy (lub moduł Freetronics EtherTen).

Szkic

W środowisku Arduino IDE należy wpisać następujący szkic, ale nie należy jeszcze wysyłać tego szkicu na płytkę:

```
/* Projekt nr 60 — budowa zdalnej stacji monitoringu
   Szkic utworzony 18 grudnia 2009 przez Davida A. Mellisa; zmodyfikowany 9 kwietnia
   2012 przez Toma Igoe;
   zmodyfikowany 20 marca 2013 przez Johna Boxalla
*/

#include <SPI.h>
#include <Ethernet.h>

❶ IPAddress ip(xxx,xxx,xxx,xxx); // W tym miejscu należy wpisać odpowiedni adres IP
❷ byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
EthernetServer server(80);

void setup()
{
   // uruchamia połączenie sieciowe i sam serwer
   Ethernet.begin(mac, ip);
   server.begin();
   for (int z=0; z<10; z++)
   {
      pinMode(z, INPUT); // ustawia piny cyfrowe nr 0 – 9 jako piny wejściowe
   }
}

void loop()
{
   // nasłuchuje żądań klientów (przychodzących żądań pobrania strony)
   EthernetClient client = server.available();
   if (client) {
      // żądanie HTTP kończy się pustym wierszem
      boolean currentLineIsBlank = true;
      while (client.connected()) {
         if (client.available()) {
            char c = client.read();
            if (c == '\n' && currentLineIsBlank) {
               client.println("HTTP/1.1 200 OK");
               client.println("Content-Type: text/html");
               client.println("Connection: close");
               client.println();
               client.println("<!DOCTYPE HTML>");
               client.println("<html>");
               // dodaje znacznik meta refresh, aby przeglądarka odświeżała stronę
               // co 5 sekund:
❸              client.println("<meta http-equiv=
               ↪\"refresh\" content=\"5\">");
               // umieszcza na stronie wartości poszczególnych wejściowych pinów
               // analogowych
               for (int analogChannel = 0; analogChannel < 6;
               ↪analogChannel++) {
                  int sensorReading = analogRead(analogChannel);
```

```
❹                client.print("stan wejściowego pinu analogowego ");
                 client.print(analogChannel);
                 client.print(" to ");
                 client.print(sensorReading);
                 client.println("<br />");
              }
              // umieszcza na stronie wartości pinów cyfrowych nr 0 – 9
              for (int digitalChannel = 0; digitalChannel < 10;
              ➥digitalChannel++)
              {
                 boolean pinStatus = digitalRead(digitalChannel);
                 client.print("pin cyfrowy nr ");
                 client.print(digitalChannel);
                 client.print(" ma wartość ");
                 client.print(pinStatus);
                 client.println("<br />");
              }
              client.println("</html>");
              break;
           }
           if (c == '\n') {
              // początek nowego wiersza
              currentLineIsBlank = true;
           }
           else if (c != '\r') {
              // otrzymano znak w bieżącym wierszu
              currentLineIsBlank = false;
           }
        }
     }
     // przeglądarka WWW potrzebuje czasu na odebranie danych
     delay(1);
     // zamyka połączenie:
     client.stop();
  }
}
```

Działanie tego szkicu zostanie bardziej szczegółowo omówione nieco później. Przed wysłaniem tego szkicu należy we wskazanym miejscu wpisać adres IP modułu sieciowego, tak aby ten moduł mógł funkcjonować w ramach sieci lokalnej lub w komunikacji z modemem. Trzy pierwsze części składowe tego adresu można określić, sprawdzając adres IP routera. Jeśli na przykład router ma adres 192.168.0.1, wystarczy zmienić ostatnią liczbę na dowolną inną (różną od cyfr przypisanych pozostałym urządzeniom w sieci — może to być dowolna jeszcze nieużywana liczba z przedziału od 1 do 254). Adres należy wpisać w wierszu ❶ szkicu:

```
IPAddress ip(192, 168, 0, 69); // adres IP modułu sieciowego
```

Po wprowadzeniu tej zmiany należy zapisać i wysłać szkic na płytkę Arduino. Musimy jeszcze umieścić moduł sieciowy na płytce Arduino, połączyć ten moduł przewodem sieciowym z routerem lub modemem i włączyć zasilanie płytki Arduino.

Po odczekaniu około 20 sekund należy otworzyć przeglądarkę internetową na dowolnym urządzeniu lub komputerze połączonym z siecią lokalną i wpisać adres IP z wiersza ❶. Jeśli na ekranie pojawi się strona podobna do tej z rysunku 19.3, możemy przyjąć, że nasz framework dla projektów zdalnej stacji monitoringu jest gotowy.

Rysunek 19.3. Wartości pinów monitorowanych przez zdalną stację wyświetlane na stronie internetowej (dostępnej z poziomu dowolnego urządzenia oferującego przeglądarkę internetową)

Rozwiązywanie problemów

Jeśli projekt nie działa, warto spróbować podjąć następujące działania:

- Należy sprawdzić, czy w wierszu ❶ szkicu wpisano prawidłowy adres IP.
- Należy sprawdzić, czy szkic jest prawidłowy i czy został wysłany na płytkę Arduino.
- Należy dwa razy sprawdzić działanie sieci lokalnej. Warto sprawdzić, czy po podłączeniu tego samego przewodu komputer dysponuje prawidłowym połączeniem z internetem. Jeśli tak, należy upewnić się, że zasilanie płytki Arduino jest włączone i że płytka jest połączona z routerem lub modemem.
- Jeśli strona internetowa generowana przez ten projekt jest otwierana za pomocą smartfonu, warto sprawdzić, czy telefon ma dostęp do lokalnej sieci Wi-Fi, nie połączenia internetowego sieci telefonii komórkowej.
- Jeśli po włączeniu zasilania płytki Arduino i połączeniu systemu z routerem lub modemem żadna z diod LED na module sieciowym nie miga, należy sprawdzić, czy projekt nie zadziała po zastosowaniu innego przewodu.

Działanie szkicu

Jeśli stacja monitoringu działa prawidłowo, możemy wrócić do analizy najważniejszych elementów szkicu. Kod od początku szkicu do wiersza ❸ jest niezbędny do aktywowania urządzeń sieciowych, załadowania potrzebnych bibliotek i uruchomienia modułu sieciowego (w funkcji `void setup`). Zastosowane przed wierszem ❸ wywołania funkcji `client.print` przygotowują stronę internetową, tak aby mogła być prawidłowo odczytywana i wyświetlana przez przeglądarkę. Począwszy od wiersza ❸, wywołania funkcji `client.print` i `client.println` służą do umieszczania informacji na stronie internetowej w taki sam sposób, w jaki wcześniej dane były wysyłane do monitora portu szeregowego. Na przykład poniższy kod powoduje wyświetlenie pierwszych sześciu wierszy na stronie (patrz rysunek 19.3):

```
client.print("stan wejściowego pinu analogowego ");
client.print(analogChannel);
client.print(" to ");
client.print(sensorReading);
```

W wierszu ❹ szkicu widać przykład umieszczania tekstu i wartości zmiennej w kodzie generowanej strony internetowej. Do określania wyglądu wyświetlanej strony można używać znaczników języka HTML — warto jednak zachować ostrożność, aby ten kod nie zajmował więcej miejsca, niż jest dostępne w pamięci systemu Arduino. Innymi słowy, możemy stosować tyle kodu HTML-a, ile będzie konieczne, ale nie więcej, niż wynosi maksymalny rozmiar szkicu, który zależy od ilości pamięci na płytce Arduino. (Rozmiary pamięci na poszczególnych wersjach płytek Arduino opisano w tabeli 11.2 w rozdziale 11.).

Warto jeszcze zwrócić uwagę na adres MAC, który jest wykorzystywany w sieciach komputerowych do wykrywania i identyfikacji poszczególnych urządzeń. Każde urządzenie sieciowe ma przypisany unikatowy adres MAC. W kodzie szkicu odpowiedni adres można ustawić, zmieniając wartości szesnastkowe w wierszu ❷. Jeśli w jednej sieci działają co najmniej dwa projekty na bazie płytki Arduino, dla każdego urządzenia należy wpisać inny adres MAC ❷.

I wreszcie jeśli generowana strona internetowa ma być dostępna na urządzeniach, które nie są połączone z lokalną siecią komputerową (na przykład na tablecie lub smartfonie uzyskującym dostęp do internetu za pośrednictwem sieci komórkowej), należy zastosować technikę nazywaną przekierowywaniem portów przy użyciu publicznego adresu IP uzyskanego od takich firm jak no-ip (*http://www.no-ip.com/*) czy Dyn (*http://dyn.com/dns/*). Metoda konfiguracji przekierowywania portów zwykle zależy od producenta i modelu routera, zatem warto poszukać w internecie szczegółowych instrukcji dla konkretnego urządzenia lub zapoznać się z podręcznikiem dostępnym na stronie *http://www.wikihow.com/Port-Forward/*.

Skoro wyjaśniłem już, jak wyświetlać tekst i wartości zmiennych na stronie internetowej, spróbujmy wykorzystać platformę Arduino do publikowania wpisów na Twitterze.

Projekt nr 61: ćwierkające Arduino

W tym projekcie pokażę, jak zbudować na bazie płytki Arduino system publikujący wpisy na Twitterze. Wszystkie informacje, które mogą być generowane przez szkic systemu Arduino, można odczytywać z poziomu dowolnego urządzenia z dostępem do Twittera. Jeśli na przykład chcemy w czasie podróży otrzymywać co godzinę informację o temperaturze w domu lub powiadomienia o powrocie dzieci ze szkoły, opisywany projekt będzie wygodnym i niedrogim rozwiązaniem.

System Arduino będzie potrzebował własnego konta na Twitterze — aby je utworzyć, należy wykonać następujące kroki:

1. Odwiedź stronę *https://twitter.com/* i utwórz nowe konto dla systemu Arduino. Zapisz sobie nazwę użytkownika i hasło.
2. Pobierz specjalny token z witryny *http://arduino-tweet.appspot.com/*, który umożliwi komunikację platformy Arduino z usługą Twitter. Na wspomnianej stronie wystarczy wykonać jeden krok.
3. Skopiuj token (wraz z danymi nowego konta na Twitterze) do pliku tekstowego na swoim komputerze.
4. Pobierz i zainstaluj bibliotekę Twitter Arduino dostępną na stronie *http://playground.arduino.cc/Code/TwitterLibrary/*.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- przewód USB;
- przewód sieciowy;
- płytka Arduino Uno i moduł sieciowy (lub moduł Freetronics EtherTen).

Szkic

W środowisku Arduino IDE należy wpisać następujący szkic, ale *nie należy jeszcze wysyłać* tego szkicu na płytkę:

```
// Projekt nr 61 — ćwierkające Arduino
#include <SPI.h>
#include <Ethernet.h>
#include <Twitter.h>

❶ byte ip[] = { xxx,xxx,xxx,xxx };
❷ byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
❸ Twitter twitter("token");

❹ char msg[] = "Żyję!"; // komunikat do publikacji na Twitterze
```

```
void setup()
{
   delay(30000);
   Ethernet.begin(mac, ip);
   Serial.begin(9600);
}

void loop()
{
   Serial.println("łączenie...");
   if (twitter.post(msg)) {
      int status = twitter.wait();
❺     if (status == 200) {
         Serial.println("OK.");
      } else {
         Serial.print("błąd, kod błędu: ");
         Serial.println(status);
      }
   } else {
      Serial.println("nie można nawiązać połączenia.");
   }
   do {} while (1);
}
```

Tak jak w projekcie nr 60, należy zdefiniować adres IP w wierszu ❶ i zmodyfikować adres MAC w wierszu ❷. W wierszu ❸ pomiędzy cudzysłowami należy umieścić uzyskany wcześniej token Twittera. I wreszcie w wierszu ❹ należy umieścić tekst, który ma być opublikowany na Twitterze. Należy teraz wysłać szkic na płytkę Arduino i podłączyć przewód sieciowy. (Warto oczywiście dodać konto systemu Arduino na Twitterze do listy kont obserwowanych przez nasze właściwe konto)! Mniej więcej po minucie należy otworzyć serwis Twitter w przeglądarce — wysłany wpis powinien już być widoczny (patrz rysunek 19.4).

Rysunek 19.4. Pierwszy wpis systemu Arduino na Twitterze

Podczas budowy systemu publikacji wpisów na Twitterze warto pamiętać o tym, że nie można wysyłać więcej niż jednego wpisu na minutę i że wszystkie wpisy muszą być unikatowe. (Wymienione ograniczenia narzuca serwis Twitter). Podczas wysyłania wpisów Twitter generuje odpowiedzi zawierające kody stanu. Szkic

odbiera i wyświetla te kody w oknie monitora portu szeregowego (wiersz ❺). Przykład wyświetlonych danych pokazano na rysunku 19.5. Komunikat z kodem 403 (jak na rysunku) oznacza, że albo zastosowany token jest nieprawidłowy, albo szkic próbuje zbyt często publikować wpisy. (Kompletną listę kodów błędów Twittera można znaleźć na stronie *http://dev.twitter.com/docs/error-codes-responses/*).

Rysunek 19.5. Wynik zbyt szybkiego wysyłania wpisów na Twittera

Sterowanie systemem Arduino za pośrednictwem strony internetowej

Systemem Arduino można sterować z poziomu przeglądarki internetowej na wiele różnych sposobów. Po zapoznaniu się z dostępnymi materiałami odkryłem metodę, która jest nie tylko niezawodna i bezpieczna, ale też darmowa — znalazłem usługę Teleduino.

Teleduino to darmowa usługa stworzona przez Nathana Kennedy'ego, entuzjastę platformy Arduino z Nowej Zelandii. To stosunkowo proste narzędzie oferuje spore możliwości w zakresie komunikacji z platformą Arduino za pośrednictwem internetu. Usługa nie wymaga żadnych specjalnych szkiców — aby sterować systemem Arduino, wystarczy wpisać specjalny adres URL w przeglądarce internetowej. Za pomocą Teleduino można na przykład sterować cyfrowymi pinami wyjściowymi i silnikami oraz wysyłać rozkazy za pośrednictwem magistrali I^2C. Co więcej, narzędzie jest stale rozwijane i uzupełniane o nowe funkcje. W projekcie nr 62 pokażę, jak skonfigurować usługę Teleduino i jak zdalnie sterować cyfrowymi pinami wyjściowymi z poziomu dowolnego urządzenia udostępniającego przeglądarkę internetową.

UWAGA *Usługa Teleduino działa tylko z płytką Arduino Uno i zgodnymi platformami zbudowanymi na bazie mikrokontrolera ATmega328.*

Projekt nr 62: konfiguracja usługi Teleduino i zdalne sterowanie systemem Arduino

Zanim będzie możliwa realizacja pierwszego projektu przy użyciu usługi Teleduino, należy się zarejestrować i uzyskać unikatowy klucz identyfikujący nasz system Arduino. W tym celu należy otworzyć stronę *https://www.teleduino.org/tools/request-key/* i wpisać niezbędne informacje. Klucz, który zostanie wysłany na wskazany adres poczty elektronicznej, będzie wyglądał mniej więcej tak: `187654321Z9AEFF952ABCDEF8534B2BBF`.

Otrzymany klucz należy następnie przekonwertować na zmienną tablicową za pomocą narzędzia dostępnego na stronie *https://www.teleduino.org/tools/arduino-sketch-key/*. Po wpisaniu klucza na stronie powinna zostać wyświetlona tablica podobna do tej pokazanej na rysunku 19.6.

```
byte key[] = { 0x65, 0x9A, 0xCE, 0xB1,
               0xA7, 0x5E, 0x57, 0x2B,
               0x8F, 0xFE, 0xA4, 0x40,
               0x9E, 0x7B, 0x7A, 0xBC };
```

Rysunek 19.6. Klucz usługi Teleduino w formie tablicy

Każdy klucz jest unikatowy i przypisany do jednej płytki Arduino, ale jeden użytkownik może uzyskać wiele kluczy, aby zrealizować więcej niż jeden projekt na bazie usługi Teleduino.

Sprzęt

Oto lista urządzeń potrzebnych do realizacji tego projektu:

- przewód USB;
- przewód sieciowy;
- płytka Arduino Uno i moduł sieciowy (lub moduł Freetronics EtherTen);
- rezystor 560 Ω (R1);
- płytka uniwersalna;
- dioda LED dowolnego koloru.

Wymienione komponenty należy połączyć w sposób pokazany na rysunku 19.7 (diodę LED należy podłączyć do pinu cyfrowego nr 8).

Szkic

Projekty korzystające z usługi Teleduino stosują tylko jeden szkic (dołączony do biblioteki Teleduino). Aby uzyskać dostęp do tego szkicu, należy wykonać następujące kroki:

Rysunek 19.7. Schemat obwodu dla projektu nr 62

1. Pobierz i zainstaluj bibliotekę Teleduino dostępną na stronie *https://www.teleduino.org/downloads/*.
2. Ponownie uruchom środowisko Arduino IDE i wybierz opcję *Plik/Przykłady/Teleduino328/TeleduinoEthernetClientProxy*.
3. W środowisku IDE zostanie wyświetlony szkic Teleduino. Przed wysłaniem tego szkicu na płytkę Arduino należy zastąpić domyślny klucz wygenerowaną wcześniej tablicą. Zmienna, którą trzeba zastąpić, powinna się znajdować w 36. wierszu szkicu. Po zastąpieniu tej zmiennej należy zapisać i wysłać szkic na płytkę Arduino.

Możemy teraz połączyć platformę Arduino z siecią i obserwować diodę LED. Po upływie około minuty dioda powinna kilka razy zamigać i zgasnąć. Liczba mignięć reprezentuje stan usługi Teleduino (patrz tabela 19.1).

Tabela 19.1. *Stan usługi Teleduino zakodowany w formie migania diody*

Liczba mignięć	Komunikat
1	Inicjalizacja
2	Nawiązywanie połączenia sieciowego
3	Łączenie z serwerem Teleduino
4	Uwierzytelnianie zakończone pomyślnie
5	Sesja już istnieje
6	Nieprawidłowy klucz lub błąd autoryzacji
10	Zerwane połączenie

Pięć mignięć diody LED oznacza, że jakaś inna płytka Arduino została już zaprogramowana przy użyciu danego klucza i nawiązała połączenie z serwerem Teleduino. Dziesięć mignięć oznacza, że należy sprawdzić komponenty sprzętowe i połączenie internetowe. Po prawidłowym nawiązaniu połączenia dioda LED powinna migać raz na około 5 sekund. Ponieważ dioda LED stanu połączenia jest sterowana przez pin cyfrowy nr 8, podczas stosowania usługi Teleduino nie możemy wykorzystać tego pinu do żadnych innych celów.

Zdalne sterowanie systemem Arduino

Do zdalnego sterowania systemem Arduino za pośrednictwem usługi Teleduino wystarczy dowolne urządzenie z przeglądarką internetową. Polecenia sterujące systemem Arduino można wydawać, wpisując odpowiednie adresy URL: *http://us01.proxy.teleduino.org/api/1.0/328.php?k=<KLUCZ>&r=setDigital--Output&pin=<X>&output=<S>*.

W powyższym adresie należy zmienić trzy parametry. Słowo *<KLUCZ>* musimy zastąpić kluczem uzyskanym wcześniej na stronie usługi Teleduino. Należy też zastąpić znaki *<X>* numerem pinu cyfrowego, którego stan ma być zmieniony. I wreszcie musimy zmienić znaki *<S>* na 0 lub LOW bądź na 1 lub HIGH, aby ustawić stan wskazanego pinu wyjściowego. Aby na przykład ustawić stan HIGH na pinie cyfrowym nr 7, należałoby użyć adresu *http://us01.proxy. teleduino.org/api/1.0/328.php?k=<KLUCZ>&r=setDigitalOutput&pin=7&output=1*.

Po prawidłowym wykonaniu tak przesłanego polecenia w przeglądarce internetowej powinien pojawić się następujący komunikat:

```
{"status":200,"message":"OK","response"
{"result":0,"time":0.22814512252808,"values":[]}}
```

Jeśli próba wykonania polecenia nie powiodła się, w oknie przeglądarki zostanie wyświetlony komunikat o błędzie:

```
{"status":403,"message":"Key is offline or invalid.","response":[]}
```

Modyfikując przywołany adres URL, można łatwo wysyłać polecenia zmiany stanu pinów cyfrowych z HIGH na LOW (i odwrotnie). Po przygotowaniu adresów URL dla projektu warto je dodać do ulubionych (zakładek) przeglądarki internetowej lub utworzyć lokalną stronę internetową z odpowiednimi linkami w formie przycisków. Możemy na przykład dodać do ulubionych stron dwa adresy URL ustawiające odpowiednio stan HIGH i LOW na pinie cyfrowym nr 7.

W pewnych sytuacjach stan pinów wyjściowych na płytce Arduino może mieć bardzo duże znaczenie. Aby zabezpieczyć projekt przed nieplanowanym zresetowaniem systemu Arduino (wskutek braku zasilania lub innych czynników), należy określić domyślny stan poszczególnych pinów cyfrowych. W tym celu trzeba otworzyć stronę *https://www.teleduino.org/tools/manage-presets/* (po nawiązaniu połączenia z usługą Teleduino). Wpisanie unikatowego klucza powinno spowodować wyświetlenie strony, na której będzie można ustawić tryby i wartości poszczególnych pinów cyfrowych (patrz rysunek 19.8).

Rysunek 19.8. Strona konfiguracji domyślnego stanu pinów

Co dalej?

Oprócz monitorowania systemu Arduino za pośrednictwem internetu i publikowania wpisów na Twitterze w tym rozdziale pokazałem, jak sterować projektami na bazie tego systemu przez internet bez konieczności przygotowywania złożonych szkiców, zdobywania zaawansowanej wiedzy na temat sieci komputerowych ani ponoszenia dodatkowych wydatków. Opisane rozwiązanie umożliwia sterowanie platformą Arduino praktycznie z każdego miejsca i — tym samym — znacznie poszerza możliwości przekazywania danych pomiędzy tym systemem a światem zewnętrznym. Trzy projekty omówione w tym rozdziale mogą stanowić dobry punkt wyjścia do planowania i budowy własnych projektów wymagających zdalnego sterowania.

W następnym, ostatnim rozdziale tej książki pokażę, jak sprawić, że system Arduino będzie wysyłał i odbierał polecenia za pośrednictwem sieci telefonii komórkowej.

20

Komunikacja w sieci telefonii komórkowej

W tym rozdziale:

- sprawisz, że system Arduino będzie dzwonił pod wskazany numer w reakcji na określone zdarzenia;
- nauczysz się wysyłać z poziomu platformy Arduino wiadomości tekstowe na telefony komórkowe;
- dowiesz się, jak sterować urządzeniami połączonymi z płytką Arduino przy użyciu wiadomości tekstowych wysyłanych za pomocą telefonu.

Projekty budowane na bazie platformy Arduino można łączyć z siecią telefonii komórkowej, aby umożliwić prostą komunikację pomiędzy tym systemem a telefonami komórkowymi i stacjonarnymi. Wystarczy odrobina wyobraźni, by wymyślić mnóstwo zastosowań dla tego rodzaju komunikacji — kilka takich przykładów zostanie omówionych w tym rozdziale.

Warto zapoznać się z całą treścią tego rozdziału przed podjęciem decyzji o zakupie dodatkowego sprzętu, ponieważ możliwość realizacji projektów w dużej mierze będzie zależała od dostępnej sieci telefonii komórkowej. Warunkiem udanej

budowy systemów opisanych w tym rozdziale jest dostęp do sieci o następujących parametrach:

- działanie w paśmie GSM 850 MHz, GSM 900 MHz, DCS 1800 MHz lub PCS 1900 MHz;
- możliwość stosowania urządzeń innych niż te oferowane przez operatora telefonii komórkowej.

Zdecydowana większość sieci telefonii komórkowej w Unii Europejskiej, Australii i Nowej Zelandii spełnia te wymagania. Czytelników ze Stanów Zjednoczonych i Kanady zachęcam do kontaktu z operatorami i potwierdzenia zgodności sieci z tymi wymaganiami przed zakupem jakiegokolwiek sprzętu.

Na potrzeby projektów opisanych w tym rozdziale warto wybrać plan przedpłacony (ang. *prepaid*) lub ofertę obejmującą dużo darmowych wiadomości tekstowych (SMS), ponieważ ewentualny błąd w kodzie szkicu może spowodować wysłanie bardzo dużej liczby takich wiadomości. Warto też sprawdzić, czy zostało wyłączone zabezpieczenie polegające na konieczności każdorazowego podawania kodu PIN przypisanego do karty SIM. (Zmiana tej opcji nie jest trudna — wystarczy umieścić kartę SIM w zwykłym telefonie komórkowym i zastosować odpowiednie ustawienia w menu poświęconym zabezpieczeniom).

Sprzęt

Z uwagi na ograniczenia urządzeń używanych do realizacji projektów opisanych w tym rozdziale można używać tylko płytki Arduino Uno lub zgodnych — oznacza to, że nie jest możliwa budowa tych urządzeń na przykład na bazie płytek Arduino Mega. We wszystkich projektach będzie stosowana ta sama konfiguracja sprzętowa, zatem warto zacząć właśnie od jej przygotowania.

Do realizacji projektów opisanych w tym rozdziale będzie potrzebny specjalny sprzęt, w tym moduł GSM SM5100 oraz antena (patrz rysunek 20.1). Moduł jest dostępny w sklepie SparkFun oraz u dystrybutorów produktów tej firmy. (Wystarczy poszukać modułu oznaczonego symbolem CEL-09607, złącza PRT-10007 oraz anteny CEL-00675).

Budowany system będzie też wymagał dodatkowego modułu zasilania i samego zasilacza. Moduł GSM pobiera prąd o natężeniu dochodzącym nawet do 2 A (to więcej, niż może zapewnić płytka Arduino), zatem bez zewnętrznego źródła zasilania może łatwo uszkodzić płytkę Arduino. Moduł zasilania zgodny z platformą Arduino (patrz rysunek 20.2) jest dostępny w sklepie DFRobot pod adresem *http://www.dfrobot.com/* (numer produktu: DFR0105).

I wreszcie będziemy potrzebowali zewnętrznego źródła zasilania. W tej roli można wykorzystać dowolny zasilacz prądu stałego (może to być nawet duży akumulator 7,2 V, panel słoneczny, bateria 12 V lub dowolne inne źródło zasilania, którego napięcie nie przekracza 35 V i które może dostarczać prąd o natężeniu dochodzącym do 2 A.

Rysunek 20.1. Moduł GSM z podłączoną anteną

Rysunek 20.2. Moduł zasilania firmy DFRobot

Przygotowanie modułu zasilania

Przed podłączeniem modułu zasilania do pozostałych komponentów należy ten moduł przygotować i — co bardzo ważne — ustawić jego napięcie wyjściowe. W tym celu musimy najpierw usunąć zworki łączące piny w prawym górnym i prawym dolnym rogu płytki (oznaczone okręgami na rysunku 20.2). Musimy też upewnić się, że dwie zworki łączą poziomo dwie pary pinów PWRIN w lewym dolnym rogu płytki (także to miejsce oznaczono na rysunku 20.2).

Możemy teraz podłączyć zewnętrzne źródło zasilania do bloku zacisków PWRIN w lewym dolnym rogu płytki modułu. Warto upewnić się, że przewody dodatni i ujemny zostały połączone z odpowiednimi złączami. Po włączeniu zasilania pozostaje nam już tylko zmierzenie napięcia na bloku zacisków PWROUT w lewym górnym rogu płytki i dostosowanie poziomu napięcia za pomocą małego niebieskiego potencjometru (patrz rysunek 20.3).

Rysunek 20.3. Potencjometr ustawiania napięcia wyjściowego modułu zasilania

Ustawienie potencjometru należy korygować do momentu, w którym napięcie zmierzone za pomocą multimetru będzie równe 5,0 V. Po ustawieniu napięcia wyjściowego należy wyłączyć moduł zasilania.

Czas połączyć ze sobą wszystkie niezbędne komponenty:

1. W gnieździe modułu GSM należy umieścić kartę SIM.
2. Moduł GSM należy zamontować na płytce Arduino Uno.
3. Na szczycie systemu należy umieścić moduł zasilania.
4. Dodatnie złącze bloku PWROUT modułu zasilania należy połączyć z pinem 5 V tego modułu; drugim przewodem trzeba połączyć ujemne złącze bloku PWROUT z pinem GND na płytce modułu zasilania. Przewody będą służyły do dostarczania prądu o napięciu 5 V i wysokim natężeniu z modułu zasilania do płytki Arduino.

OSTRZEŻENIE *Zasilanie opisanego systemu zawsze należy włączać przed połączeniem płytki Arduino z komputerem za pomocą przewodu USB. Nie należy też wyłączać zewnętrznego zasilania modułu przed odłączeniem przewodu USB.*

Konfigurowanie i testowanie sprzętu

Czas skonfigurować i przetestować opisany sprzęt, aby mieć pewność, że moduł GSM rzeczywiście może się komunikować zarówno z siecią telefonii komórkowej, jak i z płytką Arduino. Po połączeniu wszystkich urządzeń i umieszczeniu karty SIM w gnieździe modułu GSM należy wpisać i wysłać na płytkę Arduino szkic z listingu 20.1.

Listing 20.1. Szkic testujący moduł GSM

```
// Listing 20.1
// Autor: Ryan Owens — SparkFun CC v3.0 3/8/10

❶ #include <SoftwareSerial.h> // wirtualny port szeregowy
❷ SoftwareSerial cell(2,3);
  char incoming_char = 0;

  void setup()
  {
     // inicjalizuje porty szeregowe na potrzeby komunikacji:
     Serial.begin(9600);
❸    cell.begin(9600);
     Serial.println("Uruchamianie komunikacji z modułem SM5100B...");
  }

  void loop()
  {
     // jeśli otrzymano znak z modułu GSM...
     if( cell.available() > 0 )
     {
        //...należy pobrać ten znak za pośrednictwem portu szeregowego tego modułu
        incoming_char = cell.read();
        // wyświetla otrzymany znak na terminalu:
        Serial.print(incoming_char);
     }
     // jeśli otrzymano znak wysłany z terminala do płytki Arduino...
     if( Serial.available() > 0 )
     {
        incoming_char = Serial.read(); // pobiera znak wysłany z terminala
        cell.print(incoming_char); // wysyła ten znak do modułu GSM
     }
  }
```

Działanie tego szkicu sprowadza się do przekazywania wszystkich informacji otrzymywanych od modułu GSM do okna monitora portu szeregowego. Moduł GSM komunikuje się z płytką Arduino za pośrednictwem połączenia szeregowego doprowadzonego do pinów cyfrowych nr 2 i 3, zatem nie zakłóca funkcjonowania standardowego połączenia szeregowego pomiędzy płytką Arduino a komputerem PC, które wykorzystuje piny cyfrowe nr 0 i 1. Wirtualny port szeregowy dla modułu GSM skonfigurowano w wierszach ❶, ❷ i ❸. (Niezbędna biblioteka jest domyślnie instalowana wraz ze środowiskiem Arduino IDE).

Po wysłaniu szkicu na płytkę Arduino należy otworzyć okno monitora portu szeregowego i odczekać około 30 sekund. Na ekranie powinny zostać wyświetlone dane podobne do tych z rysunku 20.4.

Rysunek 20.4. Przykładowe dane wynikowe wyświetlone przez szkic z listingu 20.1

Warto zwrócić uwagę na komunikaty rozpoczynające się od przedrostka `+SIND:`, ponieważ na ich podstawie można określić status modułu GSM i jego połączenia z siecią telefonii komórkowej. Jeśli ostatnim wyświetlonym komunikatem jest `+SIND: 4`, moduł powinien dysponować prawidłowym połączeniem, zatem możemy przystąpić do realizacji projektu nr 63. Jeśli jednak ostatnim komunikatem jest `+SIND: 8`, należy zmienić częstotliwość stosowaną przez moduł GSM (patrz następny punkt).

Zmiana częstotliwości pracy

Aby zmienić częstotliwość pracy modułu GSM, należy wykonać następujące kroki:

1. Zamknij środowisko Arduino IDE i okno monitora portu szeregowego. Załaduj program terminala opisany przy okazji realizacji projektu nr 48.
2. Wybierz ten sam port COM, który jest używany przez płytkę Arduino, i kliknij przycisk *Connect*.
3. Znajdź w tabeli 20.1 częstotliwość stosowaną w sieci telefonii komórkowej i zanotuj numer pasma.
4. Wpisz rozkaz `AT+SBAND?` i naciśnij klawisz *Enter*. Moduł powinien odpowiedzieć, wyświetlając aktualne ustawienia dotyczące pasma (patrz rysunek 20.5).
5. Aby ustawić odpowiednie pasmo pracy modułu GSM, wpisz rozkaz `AT+SBAND=`*x*, gdzie *x* to numer pasma wybrany na podstawie tabeli 20.1. Moduł GSM powinien zwrócić potwierdzenie zmiany pasma (patrz rysunek 20.6).
6. I wreszcie musimy zresetować system Arduino i sprawdzić, czy prawidłowo nawiązuje połączenie z siecią telefonii komórkowej (czy wyświetla komunikat `+SIND: 4`). W tym celu połącz piny GND i RST na płytce Arduino, aby wymusić zresetowanie systemu; po chwili w oknie monitora portu szeregowego powinny pojawić się kody stanu połączenia (patrz rysunek 20.7).

Tabela 20.1. Częstotliwości i pasma pracy modułu GSM

Częstotliwość	Pasmo
GSM 900 MHz	0
DCS 1800 MHz	1
PCS 1900 MHz	2
GSM 850 MHz	3
GSM 900 MHz i DCS 1800 MHz	4
GSM 850 MHz i GSM 900 MHz	5
GSM 850 MHz i DCS 1800 MHz	6
GSM 850 MHz i PCS 1900 MHz	7
GSM 900 MHz i PCS 1900 MHz	8
GSM 850 MHz, GSM 900 MHz i DCS 1800 MHz	9
GSM 850 MHz, GSM 900 MHz i PCS 1900 MHz	10

Rysunek 20.5. Przykładowa odpowiedź modułu GSM

Rysunek 20.6. Udana zmiana pasma

Rysunek 20.7. Komunikaty sygnalizujące prawidłowe połączenie z siecią telefonii komórkowej

UWAGA *Wartość parametru* `SBAND` *wystarczy zmienić raz, ponieważ wybrane pasmo jest zapisywane w pamięci EEPROM modułu GSM. Zmiana tego parametru będzie konieczna tylko w sytuacji, gdy urządzenie będzie wykorzystywane w innym miejscu i będzie się łączyło z siecią działającą w innym paśmie.*

Czas na ostatni test — wywołanie modułu GSM za pomocą innego telefonu. Identyfikator telefonu, z którego zadzwonimy pod numer przypisany modułowi GSM, powinien zostać wyświetlony w danych wynikowych generowanych przez moduł GSM — patrz rysunek 20.8. (Widoczny na rysunku numer 0452280886 oczywiście zostanie zastąpiony numerem Twojego telefonu).

```
+CLIP: "0452280886",129

RING

+CLIP: "0452280886",129

RING
```

Rysunek 20.8. Wynik wywołania modułu GSM

Po przeprowadzeniu opisanych testów możemy być pewni, że nasze urządzenie działa zgodnie z oczekiwaniami. Czas na właściwe projekty!

Projekt nr 63: budowa dzwoniącego Arduino

Po zrealizowaniu tego projektu system Arduino będzie dzwonił pod wybrany numer telefonu w odpowiedzi na zdarzenie wykrywane przez zastosowany szkic. Jeśli na przykład temperatura w zamrażarce przekroczy określony próg lub jeśli zostanie aktywowany alarm antywłamaniowy, system może zadzwonić pod wskazany numer telefonu i zakończyć połączenie po określonym czasie (na przykład po 20 sekundach). Numer przypisany do modułu GSM pozwoli nam łatwo zidentyfikować system Arduino jako dzwoniącego i — tym samym — zinterpretować otrzymany sygnał.

Sprzęt

W tym projekcie zostanie użyty sprzęt opisany na początku tego rozdziału oraz dodatkowe komponenty niezbędne do wykrywania zdarzeń, które mają być sygnalizowane przez ten system. Aby zademonstrować działanie tego projektu, zastosujemy przycisk powodujący nawiązanie połączenia telefonicznego.

Oto lista urządzeń potrzebnych do realizacji tego projektu (oprócz sprzętu omówionego we wcześniejszej części tego rozdziału):

- jeden przycisk;
- rezystor 10 kΩ;
- kondensator 100 nF;
- różne przewody;
- płytka uniwersalna.

Schemat

Zewnętrzny obwód należy połączyć zgodnie ze schematem pokazanym na rysunku 20.9.

Rysunek 20.9. Schemat obwodu dla projektu nr 63

Szkic

W środowisku Arduino IDE należy wpisać następujący szkic, *ale nie należy go wysyłać* na płytkę Arduino:

```
// Projekt nr 63 — budowa dzwoniącego Arduino

#include <SoftwareSerial.h>
SoftwareSerial cell(2,3);

void setup()
{
   pinMode(7, INPUT);
```

```
   cell.begin(9600);
❶  delay(30000); // daje modułowi GSM czas na inicjalizację połączenia
}

❷  void callSomeone()
{
❸  cell.println("ATDxxxxxxxxxx"); // dzwoni pod numer telefonu xxxxxxxxxx
   // znaki xxxxxxxxxx należy zastąpić właściwym numerem telefonu (z numerem
   // kierunkowym)
   delay(20000); // czeka 20 sekund
   cell.println("ATH"); // koniec połączenia
   delay(60000); // czeka 60 sekund na moduł GSM
}

void loop()
{
   if (digitalRead(7) == HIGH)
   {
❹    callSomeone();
   }
}
```

Działanie szkicu

Moduł GSM jest aktywowany w kodzie funkcji `void setup`, natomiast w wierszu ❶ szkic daje temu modułowi czas na lokalizację sieci telefonii komórkowej i rejestrację w tej sieci. W roli monitorowanego zdarzenia wykorzystano przycisk podłączony do pinu cyfrowego nr 7 na płytce Arduino. Naciśnięcie tego przycisku powoduje wywołanie funkcji `callSomeone` zdefiniowanej w wierszu ❷. W wierszu ❸ szkic wysyła do modułu GSM rozkaz nawiązania połączenia ze wskazanym numerem telefonu.

Zapisane w kodzie znaki *xxxxxxxxxx* należy zastąpić numerem telefonu, pod który system Arduino ma dzwonić. Sposób definiowania numeru nie różni się od jego wpisywania w telefonie komórkowym. Jeśli na przykład system Arduino ma dzwonić pod numer +48 212 555 212, należy zastosować następujące wywołanie:

```
cell.println("ATD48212555212");
```

Po wpisaniu numeru telefonu można wysłać szkic na płytkę Arduino, odczekać minutę i przystąpić do testów systemu (w tym przypadku polegających na naciskaniu przycisku). Integracja kodu odpowiedzialnego za nawiązywanie połączeń telefonicznych z innymi istniejącymi szkicami jest bardzo prosta — wystarczy umieścić w odpowiednim miejscu szkicu takie wywołanie funkcji jak w wierszu ❹. Od tej pory to do Ciebie należy wybór zdarzeń, które powinny powodować, że system Arduino będzie dzwonił pod wskazany numer.

Spróbujmy teraz wprowadzić platformę Arduino w świat XXI wieku, w którym ważną formą komunikacji jest wysyłanie wiadomości SMS.

Projekt nr 64: budowa systemu Arduino wysyłającego SMS-y

W tym projekcie system Arduino będzie wysyłał wiadomości tekstowe na telefon komórkowy w odpowiedzi na określone zdarzenie. Aby uprościć kod, zastosujemy bibliotekę SerialGSM dla platformy Arduino (dostępną pod adresem *https://github.com/meirm/SerialGSM/*). Po zainstalowaniu tej biblioteki należy ponownie uruchomić środowisko Arduino IDE.

W tym projekcie zostanie użyty ten sam sprzęt co w projekcie nr 63.

Szkic

W środowisku Arduino IDE należy wpisać następujący szkic, ale nie należy jeszcze wysyłać tego szkicu na płytkę:

```
// Projekt nr 64 — budowa systemu Arduino wysyłającego SMS-y

#include <SerialGSM.h>
#include <SoftwareSerial.h>
❶ SerialGSM cell(2,3);

void setup()
{
   pinMode(7, INPUT);
   delay(30000); // czeka na moduł GSM
   cell.begin(9600);
}

void textSomeone()
{
   cell.Verbose(true); // ułatwia diagnozowanie szkicu
   cell.Boot();
   cell.FwdSMS2Serial();
❷  cell.Rcpt("+xxxxxxxxxxx"); // znaki xxxxxxxxxxx należy zastąpić
                              // numerem telefonu odbiorcy
❸  cell.Message("To jest treść wiadomości SMS");
   cell.SendSMS();
}

void loop()
{
❹  if (digitalRead(7) == HIGH)
   {
      textSomeone();
   }

   if (cell.ReceiveSMS())
   {
```

```
        Serial.println(cell.Message());
        cell.DeleteAllSMS();
    }
}
```

Działanie szkicu

Moduł GSM został skonfigurowany w tradycyjny sposób w wierszu ❶ i w kodzie funkcji `void setup()`. Zdarzenie naciśnięcia przycisku jest wykrywane w wierszu ❹ i powoduje wywołanie funkcji `textSomeone`. Działanie tej funkcji sprowadza się do wysłania wiadomości SMS pod numer telefonu zapisany w wierszu ❷.

Przed wysłaniem szkicu należy zastąpić znaki *xxxxxxxxxxx* numerem telefonu odbiorcy w międzynarodowym formacie obejmującym numer kierunkowy (bez spacji, nawiasów i innych znaków). Aby na przykład wysłać wiadomość pod numer 212.555.1212 w Stanach Zjednoczonych, należy zapisać w kodzie wartość +12125551212.

Tekst wysyłanej wiadomości zapisano w wierszu ❸. (Warto pamiętać, że długość wiadomości nie może przekraczać 160 znaków).

Po wpisaniu przykładowej wiadomości tekstowej i docelowego numeru telefonu można wysłać szkic na płytkę Arduino, odczekać 30 sekund i nacisnąć przycisk. Po chwili wiadomość SMS powinna dotrzeć do docelowego telefonu (patrz rysunek 20.10).

Rysunek 20.10. Przykładowa wiadomość SMS po odebraniu na telefonie

Projekt nr 64 można łatwo zintegrować z pozostałymi szkicami, a same wiadomości tekstowe mogą być uzależniane od stanu systemu (wystarczy porównywać parametry za pomocą wyrażenia `switch-case`).

UWAGA *Musimy pamiętać, że duża liczba wysyłanych wiadomości tekstowych może rodzić poważne koszty, zatem na czas eksperymentów warto zadbać albo o plan taryfowy z nieograniczoną liczbą SMS-ów, albo o plan przedpłacony.*

Projekt nr 65: konfiguracja systemu sterowanego za pomocą wiadomości SMS

W tym projekcie będziemy sterowali cyfrowymi pinami wyjściowymi na płytce Arduino poprzez wysyłanie wiadomości tekstowych z telefonu komórkowego. Na podstawie materiału zawartego w poprzednich rozdziałach możesz bez trudu dodawać różne urządzenia, którymi będziesz sterował w prezentowany tutaj sposób. W tym projekcie będziemy sterowali czterema wyjściami cyfrowymi, ale w razie potrzeby można sterować mniejszą lub większą liczbą pinów.

Aby włączyć lub wyłączyć cztery wyjścia cyfrowe (w tym przypadku piny od 10. do 13.), wystarczy wysłać do systemu Arduino wiadomość tekstową w następującym formacie: `#axbxcxdx`, gdzie *x* należy zastąpić cyfrą `0` (aby wyłączyć) lub `1` (aby włączyć). Żeby na przykład włączyć wszystkie cztery wyjścia cyfrowe, trzeba wysłać wiadomość `#a1b1c1d1`.

Sprzęt

W tym projekcie zostanie użyty sprzęt opisany na początku tego rozdziału oraz dodatkowe komponenty wybrane przez Ciebie. Sam zastosuję cztery diody LED reprezentujące stan sterowanych wyjść cyfrowych. Oznacza to, że będę potrzebował następujących komponentów:

- czterech diod LED;
- czterech rezystorów 560 Ω;
- różnych przewodów;
- płytki uniwersalnej.

Schemat

Zewnętrzny obwód należy połączyć zgodnie ze schematem pokazanym na rysunku 20.11.

Szkic

W środowisku Arduino IDE należy wpisać i wysłać następujący szkic:

```
// Projekt nr 65 — konfiguracja systemu sterowanego za pomocą wiadomości SMS

#include <SoftwareSerial.h>
SoftwareSerial cell(2,3);
char inchar;

void setup()
{
   // konfiguruje wyjściowe piny cyfrowe, którymi będziemy sterować
   pinMode(10, OUTPUT);
```

Rysunek 20.11. Obwód dla projektu nr 65

```
    pinMode(11, OUTPUT);
    pinMode(12, OUTPUT);
    pinMode(13, OUTPUT);

    digitalWrite(10, LOW); // domyślny stan pinów wejścia-wyjścia po włączeniu
                           // zasilania lub zresetowaniu systemu
    digitalWrite(11, LOW); // domyślny stan można oczywiście zmienić zależnie od
                           // potrzeb
    digitalWrite(12, LOW);
    digitalWrite(13, LOW);

// inicjalizuje port szeregowy modułu GSM na potrzeby komunikacji
    cell.begin(9600);
    delay(30000);
❶   cell.println("AT+CMGF=1");
    delay(200);
❷   cell.println("AT+CNMI=3,3,0,0");
    delay(200);
}

void loop()
{
    // jeśli otrzymano znak z modułu GSM...
```

```
❸     if(cell.available() > 0)
      {
         inchar = cell.read();
❹           if (inchar == '#') // początek polecenia
         {
            delay(10);
            inchar = cell.read();
❺           if (inchar == 'a')
            {
               delay(10);
               inchar = cell.read();
               if (inchar == '0')
               {
                  digitalWrite(10, LOW);
               }
               else if (inchar == '1')
               {
                  digitalWrite(10, HIGH);
               }
               delay(10);
               inchar = cell.read();
               if (inchar == 'b')
               {
                  inchar = cell.read();
                  if (inchar == '0')
                  {
                     digitalWrite(11, LOW);
                  }
                  else if (inchar == '1')
                  {
                     digitalWrite(11, HIGH);
                  }
                  delay(10);
                  inchar = cell.read();
                  if (inchar == 'c')
                  {
                     inchar = cell.read();
                     if (inchar == '0')
                     {
                        digitalWrite(12, LOW);
                     }
                     else if (inchar == '1')
                     {
                        digitalWrite(12, HIGH);
                     }
                     delay(10);
                     inchar = cell.read();
                     if (inchar == 'd')
                     {
                        delay(10);
                        inchar = cell.read();
```

```
                    if (inchar == '0')
                    {
                       digitalWrite(13, LOW);
                    }
                    else if (inchar == '1')
                    {
                       digitalWrite(13, HIGH);
                    }
                    delay(10);
                 }
              }
              cell.println("AT+CMGD=1,4"); // usuwa wszystkie wiadomości SMS
           }
        }
     }
  }
}
```

Działanie szkicu

W tym projekcie system Arduino monitoruje wszystkie znaki otrzymywane od modułu GSM. W wierszu ❶ wymuszamy na module GSM konwersję przychodzących wiadomości SMS na tekst. W wierszu ❷ określamy, że treść wiadomości ma być wysyłana do wirtualnego portu szeregowego. W wierszu ❸ system Arduino oczekuje na wiadomości tekstowe przesłane przez moduł GSM.

Ponieważ polecenia sterowania pinami cyfrowymi na płytce Arduino (wysyłane z telefonu i przekazywane przez moduł GSM) rozpoczynają się od znaku #, w wierszu ❹ szkic czeka na wiadomość rozpoczynającą się właśnie od tego znaku. W wierszu ❺ szkic sprawdza pierwszy parametr pinów wyjściowych — jeśli po literze a występuje cyfra 0 lub 1, pin jest odpowiednio wyłączany lub włączany. Ten sam proces jest powtarzany dla trzech kolejnych pinów (i liter b, c oraz d).

Nietrudno sobie wyobrazić, jak łatwo można przystosować ten projekt do zdalnego sterowania najróżniejszymi urządzeniami, jak światła, pompy, alarmy itp.

Co dalej?

Trzy projekty opisane w tym rozdziale stanowią doskonały punkt wyjścia do budowy własnych projektów, które wykorzystają do komunikacji sieć telefonii komórkowej. Jedynym ograniczeniem jest nasza wyobraźnia — możemy na przykład stworzyć system wysyłający wiadomość SMS w razie zalania piwnicy lub włączający klimatyzację w domu po otrzymaniu wiadomości wysłanej z telefonu komórkowego. Warto jednak raz jeszcze podkreślić, że systemy komunikujące się za pośrednictwem sieci telefonii komórkowej mogą znacznie podnieść rachunki wysyłane przez operatora GSM.

Teraz, po przeczytaniu o 65 projektach (mam nadzieję, że także po ich zrealizowaniu) powinieneś rozumieć i wiedzieć, jak budować własne projekty na bazie platformy Arduino. Wyjaśniłem działanie podstawowych elementów składowych używanych w wielu projektach i jestem pewien, że przekazana wiedza wystarczy do stosowania tej technologii podczas rozwiązywania najróżniejszych problemów. Budowa własnych projektów to doskonała zabawa.

Ta książka to dopiero początek. Istnieje wiele form i konstrukcji sprzętowych, które można łączyć na różne sposoby. Wystarczy odrobina analizy i planowania, aby wszystkie te rozwiązania skutecznie łączyć z systemem Arduino. Podczas realizacji tego celu sporym ułatwieniem może być kontakt z internetową społecznością użytkowników platformy Arduino (na przykład na oficjalnym forum tego systemu pod adresem *http://arduino.cc/forum/*) lub z hobbystami gotowymi do dzielenia się swoimi doświadczeniami i mieszkającymi w okolicy.

Nie stój w miejscu — zbuduj coś!

Skorowidz

A

A, 53
AC, 53
adres
  IP, 387, 380
    modułu sieciowego, 383
    routera, 383
    statyczny, 381
  MAC, 385, 387
  modułu, 365
alarm antywłamaniowy, 400
algorytm, 52, 82
alternating current, 53
alternatywa, 91
  bitowa, 162
  wykluczająca, 163
amper, 53
analog reference, 103
analogowy
  sygnał, 98
analogRead(), 103
analogWrite, 67, 68
animacje, 166
anoda, 57, 69
  symbol, 75
antena, 394
Arduino
  LilyPad, 249, 251
  Mega, 249
  Mega 2560, 252
  Nano, 249, 251
Arduino, 19, 37
  Due, 248, 253
  LilyPad, 249, 251
  Mega, 249, 252
  Nano, 249, 251
  piny, 242
  płytka, 20, 24
  płytki alternatywne, 247
  połączenia platformy z komputerem, 25
  programowanie, 25
  przechowywanie danych, 336
  rozszerzanie, 41, 188
  sterowanie z poziomu przeglądarki internetowej, 388
  sterowniki interfejsu USB, 31
  symbol płytki, 75
  szkice platformy, 42
  środowisko programowania, 42
  Uno, 28, 386, 389, 394
  własne płytki, 238
  zdalne sterowanie systemem, 391
Arduino.app, 27
AREF, 103
  tłumienie zakłóceń na pinie, 104
array, 146
arytmetyka bitowa, 161
ATmega2560, 248, 252
ATmega328, 336
ATmega328P, 248
  SMD, 248
attachInterrupt, 214

B

B, 69, 76, 138, 262
bajt, 139
Banzi Massimo, 19
baza, 69
  symbol, 76

BCD, 365
bezwładność wzroku, 165
biblioteka, 196
  dołączenie, 180
  I²C, 365
  instalowanie, 196
  Mac OS X, 197
  moduł, 196
  SerialGSM, 403
  servo, 258
  TinyGPS, 293
  Twitter Arduino, 386
  Ubuntu Linux, 199
  VirtualWire, 305
  Windows, 198
  Wire, 344
BIN, 138
binarne
  systemy, 137
  zmienne, 161
binarycoded decimal, 365
bit, 90
bity (porównywanie) 162
Blu-Tack, 191
błąd
  kompilacji, 49
  w kodzie (znajdowanie), 123
Boarduino, 250
boolean variable, 90
bouncing, 81
brama sieciowa routera, 380
broadboard, 58
brzęczyk
  piezoelektryczny, 106
  symbol, 107
bufor portu szeregowego, 125
  czyszczenie, 127
byte, 137, 161

C

C, 69, 76, 262
calculateservo, 261
CAT5E, 380
CAT6, 380
cel projektu, 52
cewka (symbol), 76
CHANGE, 213
chip
  resistors, 56
  select, 353
circuit diagrams, 74
clock, 141
COM, 76
common, 76
Cooper Tyler, 21
crystal oscillator, 240
CS, 353
Cuartielles David, 19
current, 53
cyfrowy
  sygnał, 98
  rezystor nastawnego, 356
czas
  pracy, 371
  wykonywania szkicu, 208
częstotliwość
  125 kHz, 331
  433 MHz, 304
  migania diody LED, 50
  pracy modułu GSM, 398
czołg, 265
czujnik
  odległości na podczerwień, 277
  temperatury, 109
czyszczenie bufora portu szeregowego, 127

D

dane
  rejestrowanie, 201
  tekstowe (przesyłanie), 304
  wysłanie do monitora portu szeregowego, 126
  wysłanie z monitora portu szeregowego, 125, 126
DC, 53
DEC, 138
decimal point, 149
delay, 48, 50, 62
DFR0105, 394
diagram połączeń, 63
digital storage oscilloscope, 81
digitalWrite, 62
dioda
  LED, 57, 61
    symbol, 76
  napięcie przewodzenia, 58
  oznaczenie, 69
  prostownicza, 69
    symbol, 75
direct current, 53
DLINE, 291, 312
długość wiadomości SMS, 404

dodawanie, 102
urządzeń, 41
domyślny stan pinów cyfrowych, 392
dostosowanie poziomu napięcia, 396
do-while(), 125
DP, 149
drganie styków, 81, 218
zabezpieczenie, 83
drukowanie szkicu, 43
DS3232, 361
DSO, 81
Duemilanove, 250
dwójkowy system, 137
dzielenie, 102
napięcia, 104

E

E, 69, 76, 262
EEPROM, 336
zapisanie wartości w pamięci, 336
zewnętrzny, 346
żywotność pamięci, 336
EEPROM.write, 336
efekty wizualne, 166
EIA-96 code calculator, 56
ekran dotykowy, 226
budowa, 227
łączenie, 226
mapa, 229, 234
przełącznik, 230
testowanie, 229, 233
ekspander portów, 350, 352
ELB149C5M, 331
electrically erasable programmable read-only memory, 336
elektroniczna kostka do gry, 133
elektryczność, 53
statyczna (pomiar), 132
element, 146
EM406, 290
emiter, 69
symbol, 76
error, 49
EtherMega, 253
EtherTen, 380, 386, 389
expected, 49
Explorer, 313

F

FALLING, 213
false, 90
fałsz, 90
farad, 78
feature creep, 52
float, 102
float voltage, 100
for, 65
Freeduino, 250
Freetronics
Eleven, 249
LCD & Keypad, 293
Fritzing, 77
FTDI, 246
funkcja, 113
zwracająca wartość, 116

G

generowanie sygnałów elektrycznych, 46, 47
GLCD.
CursorTo, 180
DrawCircle, 182
DrawHoriLine, 182
DrawRect, 182
DrawRoundRect, 182
DrawVertLine, 182
FillRect, 182
Puts, 180
SetDot, 182
global positioning system, 290
GND, 53, 77
gniazdo
Ethernet, 380
kart pamięci microSD, 380
GP2Y0A21YK0F, 277
GPS, 188, 290
dane o godzinie, 296
odbiornik, 290
stan odbiornika, 292
szybkość przekazywania danych przez odbiornik, 292
testowanie, 291
GPS-09123, 291
Greenwich Mean Time, 297
ground, 53
GSM
częstotliwość pracy, 398
status modułu, 398

GSM
test, 400
zasilanie, 394
rozkaz nawiązania połączenia, 402

H

HD44780, 172
hexTronik HXT900, 257
HIGH, 48, 62

I

I, 59
I/O, 40
I²C, 343, 344, 350
adres magistrali, 350
IDE, 25, 42
konfiguracja, 27
if, 88
if-then, 220
ikona
Monitor portu szeregowego, 44
Nowy, 44
Otwórz, 44
Weryfikuj, 44
Załaduj, 44
Zapisz, 44
ilość pamięci dostępnej, 49
infrared, 319
INPUT, 47
instrukcja
powtarzanie, 124
wykonywane jednorazowe, 46
int, 64, 161
integer, 64
integrated development environment, 25
interfejs sieci internetowej (Ethernet), 41
inter-integrated circuit, 343
interrupt, 213, 214
IPAddress ip, 384
IR, 319
iteracje, 65
izolacja od obwodu sterującego, 70

J

jasność świecenia diody, 233
jednostki miar (konwersje), 79

K

karta
dostępu, 329
microSD, 201
pamici
wpisywanie danych, 202
formatowanie, 201
katoda, 57, 69
symbol, 75
klawiatura numeryczna, 218
program obsługi, 219
testowanie, 219
kod 403, 388
modyfikowanie, 50
paskowy, 54
kolektor, 69
symbol, 76
komentarz, 45
kompilowanie, 49
komponenty
wybór, 52
komunikacja z Arduino
za pośrednictwem internetu, 388
komunikaty środowiska, 44
kondensator, 78
ceramiczny, 79
elektrolityczny, 80
rozładowanie, 78
symbol, 79
koniunkcja, 91
bitowa, 162
konwersja,
jednostek miar, 79
liczby binarnej na dziesiętną, 137
kostka do gry, 133
KS0066, 172
KS0108B, 178
k , 54

L

L, 41
latch, 141
LCD, 171
liczba w systemie
binarnym, 175
dziesiętnym, 175
szesnastkowym, 175
moduł graficzny, 178
pozycja kursora, 174

własne znaki, 176
wyświetlenie
słowa, 175
zmiennej, 175
wyzerowanie, 174
znakowy
czarna ramka z białym wypełnieniem, 182
koło, 182
liczba całkowita, 180
odcinek , 182
prostokąt z zaokrąglonymi narożnikami, 182
tekst, 180
włączenie piksela, 182
wypełniony prostokąt, 182
lcd.
begin, 174
clear, 174
print, 175
setCursor, 174
least significant bit, 137
LED, 40, 57, 61
napięcie przewodzenia, 58
poziom jasności, 66
sterowanie SMS-em, 405
symbol, 76
świecenie, 48
wyłączenie, 48
liczba
binarna 137, 140
całkowita, 64, 127
iteracji, 65
losowa, 132, 143
obrotów wału na minutę, 261
wyświetlanie, 149
light-emitting diodes, 40
Lindsay Philip, 20
linia
danych, 344
zegara, 344
liniowy
potencjometr, 105
regulator napięcia, 239
Linux, 25
Ubuntu, 33
liquid crystal display, 171
logarytmiczny potencjometr, 105
long, 127
loop, 47
LOW, 48, 62, 213
LSB, 137
LSBFIRST, 150
lutowanie, 194

Ł

ładowanie szkicu, 43
ładunek elektryczny, 78

M

MAC adres, 385
Mac OS X, 25
biblioteka, 197
magistrala
danych, 343
I²C, 344
SPI, 353
mapa ekranu dotykowego, 229
masa klejąca wielokrotnego użytku, 191
master-in, slave-out, 353
master-out, slave-in, 353
MCP23017, 350, 351, 352
MCP4162, 356
microSD, 188, 201
testowanie, 201
zapisywanie danych, 202
microSDHC, 201
microswitch, 273
miernik uniwersalny, 56
mierzenie
napięcia, 56
natężenia prądu, 56
oporu elektrycznego, 56
miganie
diody, 45
dostrzegalne, 66
mikrofarad, 78
mikrokontroler, 39, 41, 240
wyjmowanie, 244
mikroprzełącznik, 273
miliamper, 53
millis, 208
miniaturowy rezystor nastawny, 106
MISO, 353
mnożenie, 102
moc, 53
znamionowa, 56
model mostu z sygnalizatorami, 92
modem, 380
modulacja szerokości impulsu, 66, 108, 236, 271
modulo, 155

moduł, 41, 188
adres, 365
biblioteki, 196
dodatkowe, 190
GSM
częstotliwość pracy, 398
zasilanie, 394
SM5100, 394
komunikacji radiowej, 310
niestandardowy, 191
radiowy, 304
samodzielne konstruowanie, 190
sieciowy, 379
zasilania (przygotowanie), 395
modyfikowanie kodu, 50
monitor portu szeregowego, 120, 143, 349
monitorowanie
stanu pinów cyfrowych, 195
zdalne, 381
montaż wielu modułów, 190
MOSI, 353
most significant bit, 137, 354
MSB, 137, 354
MSBFIRST, 153
multimetr, 56
myservo.write, 258
M , 54

N

najbardziej znaczący bit, 137, 354
najmniej znaczący bit, 137
napięcie, 53, 59
dzielenie, 104
referencyjne, 103
regulator, 239
skok, 71
stabilizacja, 80
wewnętrzne, 104
natężenie prądu, 53, 59
ograniczanie, 54
nazwa
pliku (szkic), 43
stosowanej płytki, 44
NC, 76
negacja, 90
bitowa, 163, 164
Nintendo DS, 226
Niski stan, 99
NO, 76
noInterrupts, 214
normally
closed, 76
open, 76
NPN, 76
numer
portu
cyfrowego, 47
USB, 44
telefonu odbiorcy SMS, 404

O

obraz
odwracanie, 168
wyświetlanie, 167
obsługa przerwań, 214
obszar
komunikatów, 42, 44
poleceń, 42, 43
tekstu, 42, 44
obwód
filtrujący, 83
ochrona, 71
własny, 237
odbiornik
podczerwieni, 320
testowanie, 321
odejmowanie, 102
odwracanie obrazu, 168
OE, 141
określanie
godziny, 290
położenia, 290
przybliżonej szybkości, 290
om, 54
Oomlout, 22
open source, 254
operator
alternatywy, 91
koniunkcji, 91
negacji, 90
porównania, 103
opornik, 54
oporność, 54,59
oprogramowanie, 42
oscyloskop cyfrowy z pamięcią, 81
OUTPUT, 47, 61
output enable, 141

P

pamięć
  dostępna, 49
  EEPROM, 248
  flash, 248
  SRAM, 248
Parallax Ping, 282
Pasmo (aktualne ustawienia), 398
persistence of vision, 165
pętla for, 65
pF, 78
piezoelektryczny brzęczyk, 106
pikofarad, 78
pilot, 321
pin
  cyfrowy, 46
    stan domyślny, 392
  zwiększanie liczby, 138
numer 1, 241
pinMode, 47, 61
plan przedpłacony, 394
planowanie działań, 52
płytka
  prototypowa, 190
  uniwersalna, 58
PNP, 76
pobieranie danych ze zdalnego czujnika, 314
podczerwień, 319
podłączanie dodatkowych płytek, 41
Pololu RP5, 265
połączenia (symbol), 77
położenie (rejestrowanie), 298
pomiar (precyzja), 103
ponowne uruchamianie systemu, 41
porównywanie bitów, 162
port
  szeregowy, 40, 44
    monitor, 120
  USB, 380
potencjometr, 105
POV, 165
PowerSwitch Tail, 112
powiadomienia o powrocie dzieci ze szkoły, 386
powtarzanie instrukcji, 124
poziom jasności diod LED, 66
prawda, 90
prawo Ohma, 58, 59
prąd
  błądzący, 71, 265
  maksymalne natężenie, 68
  natężenie, 53
  przełączanie, 69
  sieciowy (przełączanie), 72
  stały, 53
  zmienny, 53
precyzyjny zegar, 296
prepaid, 394
prędkość podróży, 298
program (wstrzymanie wykonywania), 213
projekt (cel), 52
projektowania systemów, 52
ProtoScrewShield, 367
ProtoShield, 190
  RESET, 194
  układ otworów, 192
  projekt obwodu, 193
przegrzanie rezystora, 56
przekazywanie informacji pomiędzy płytką a komputerem, 50
przekaźnik, 70, 71
  symbol, 76
  zastosowanie, 71
przekierowanie portu przy użyciu publicznego adresu IP, 386
przełączanie prądu, 69, 72
przełącznik dotykowy, 233
przepustowość, 121
przerost funkcji, 52
przerwanie, 213
  tryb, 213
przesunięcie
  w lewo, 163
  w prawo, 163
przesyłanie danych tekstowych, 304
przetwarzanie dużej liczby powiązanych danych, 146
przewody, 60
  symbol, 77
przewód sieciowy, 380
przycisk symbol, 82
pull-
  down resistor, 83
  width modulation, 66
PWM, 66, 236
PWRIN, 396
PWROUT, 396

Q

quiz, 143

R

R, 59
radio-frequency identification, 329
random, 132
randomSeed, 132
readKeypad, 223
reakcja na zdarzenie, 213
real-time clock, 361
reference voltage, 103
regulator napięcia, 239
rejestr przesuwający, 138
  zasada działania, 139
rejestrowanie
  danych, 201
  położenia, 298
  ruchu samochodu, 298
RESET, 39, 41
  ProtoShield, 194
reszta z operacji dzielenia, 155
revolutions per minute, 261
rezonator kwarcowy, 240
rezystancja, 54
rezystor, 54, 61
  do montażu powierzchniowego, 56
  miniaturowy, 106
  moc znamionowa, 56
  nastawny cyfrowy, 356
  nastawny, 105, 356
  ograniczający natężenie prądu, 58
  przegrzanie, 56
  symbol, 75
  ściągający, 83
RFID, 329
  budowa, 330
  czytnik, 331
  etykiety, 331
  identyfikator etykiety, 336
  system kontroli dostępu, 333
  testowanie, 331, 332
RFR103B2B, 331
RISING, 213
robot
  czujnik podczerwieni do wykrywania kolizji, 280
  gąsienicowy, 265
  możliwe ruchy, 271
  zderzenia z obiektami, 273
rozmiar tablicy, 146
RPM, 261
RTC, 361
RTL-10709, 290
ruch wahadłowy, 92
RX, 41, 50
rzut kostką do gry, 134

S

S, 82
SAM3X8E, 248
Sarik John, 22
SBAND, 399
schemat obwodu, 74
schematic diagrams, 74
Schulz Kurt, 21
SCK, 353
SCL, 344, 362
Scooterputer, 21
SDA, 344, 362
seed, 132
Serial.
peripheral interface, 343
  available(), 126
  begin, 121
  flush(), 127
  GSM, 403
  print(), 138
  println, 122
servo, 256
servomechanism, 256
serwomotor, 256
shields, 188
shift register, 138
shiftOut(), 150
siedmiosegmentowy wyświetlacz, 148
silnik, 257
  elektryczny, 261
  dodatkowe źródła zasilania, 263
  napięcie zasilania, 261
  prąd
    bez obciążenia, 261
    zatrzymania, 261, 263
  sterowanie, 262, 264
  szybkość przy napięciu zasilacza, 261
silnik
  wykonawczy, 256
    łączenie, 257
    natężenie pobieranego prądu, 256
    szybkość, 256
    uruchamianie, 257
    wymuszenie ruchu, 258
    zakres obrotu, 256
SIM, 394

sketch, 37
skok napięcia, 71
slave select, 353
SM5100, 394
smartfon, 385
SMS, 394
  długość wiadomości, 404
  w odpowiedzi na zdarzenie, 402
SPI, 343, 353
  połączenie, 353
sprzętu wybór, 52
SS, 353
stabilizacja napięcia, 80
stan
  niski, 99
  wysoki, 48, 99
status modułu GSM, 398
sterowanie
  drogą radiową, 305
  SMS-em, 405
  urządzeniami zasilanymi sieci, 112
  za pomocą pilota na podczerwień, 323
stoper, 210
strona internetowa, 381
  z linkami w formie przycisków, 392
styki
  drganie, 81
  przekaźnika (symbol), 76
Sudoku, 22
switch bounce, 81
switch-case, 220
sygnalizacja świetlna, 92
sygnał
  analogowy, 98
  cyfrowy, 98
symbol uśmiechniętej buzi, 176
system
  binarny, 137, 365
  kontroli dostępu, 329
  mierzenia czasu pracy, 371
  nawigacji satelitarnej, 290
szerokość i długość geograficzna, 295
szkic, 42, 45, 52, 87, 94, 114, 134
  czas wykonywania, 208
  debugowanie, 123
  drukowanie, 43
  elastyczność, 64
  komentarz, 45
  kopiowanie, 43
  ładowanie, 43
  maksymalny rozmiar, 385
  port USB, 43
  przeszukiwanie, 43
  testowy, 244
  uruchamianie, 63
  weryfikacja, 43, 49
  wklejanie, 43
  wybór typu płytki, 43
  zapisywanie, 43
szybkość działania mikrokontrolera, 240

Ś

środowisko programowania, 42
  wersja, 43

T

tablica, 146
  odczytywanie, 147
  rozmiar, 146
  zapisywanie, 147
tekst
  wysłanie do monitora portu szeregowego, 121
Teleduino, 388
  klucz, 391
  konfiguracja, 389
  liczba mignięć LED, 391
  stan usługi, 391
temperatura
  pomiar, 108
  rejestrowanie, 205
  w zamrażarce, 400
Terminal, 313
termometr, 117
  analogowy, 259
  cyfrowy, 156
  LCD znakowy, 183
  zdalnie sterowany, 314
tester baterii, 99
TinyGPS, 300
TIP120, 262
TMP36, 315
tolerancja rezystora, 55
transmisja bezprzewodowa, 303
tranzystor, 68, 71
  Darlingtona, 262
  NPN, 76
  PNP, 76
  symbol, 76
  wyłączenie, 69
trimpot, 106, 172
true, 90

tryb danych wyjściowych, 47
trymer, 106
TSOP4138, 321
TWI, 344
Twitter, 22
  token, 387
  wpisy, 386
  wymagania na wpisy, 388
two wire interface, 344
TwypeWriter, 22
TX, 41, 50

U

U, 59
UART, 291, 292, 312
Ubuntu Linux, 33
  biblioteka, 199
ultradźwiękowy czujnik odległości, 282
urządzenia
  dodawanie, 41
  nadrzędne, 344
  podrzędne, 344
USB, 25, 39, 40
ustawianie daty i godziny w układzie zegara
  czasu rzeczywistego, 362
UTC, 297
uziemienie, 53
  symbol, 77

V

V, 53
variables, 64
VirtualWire, 305
void
  loop, 62, 204
  setup, 46, 47, 62
voltage, 53

W

W, 53
W5100, 379
wartość początkowa, 132
warunek
  sprawdzanie, 124
    po wykonaniu kodu, 125
wat, 53
wejścia
  analogowe, 39
  -wyjście, 40
wersja środowiska IDE, 43
weryfikacja szkicu, 49
wewnętrzne napięcie referencyjne, 104
while(), 124
wielokrotne wykonanie instrukcji, 47
Wi-Fi, 385
Windows, 8, 25, 29
  biblioteka, 198
Windows XP, 25
Wire, 344
WLS107B4B, 304
własny obwód, 237
włączanie klimatyzacji SMS-em, 408
wolt, 53
wpisy na Twitterze, 386
WRL-08687, 311
WRL-10534, 304
wspólna
  anoda, 149
  katoda, 149
współczynnik wypełnienia impulsu, 66
wstrzymanie wykonywania
  programu, 213
  szkicu, 48
wykrywanie sygnałów elektrycznych, 46, 47
wysoki stan, 99
wyświetlacz, 21
  LED (sterowanie), 150
  liczb binarnych, 140
  matrycowy, 158
  siedmiosegmentowy, 148
wyświetlanie
  daty i godziny na sznakowym module LCD, 367
  na module LCD współrzędnych
    reprezentujących bieżącą pozycję, 293
wzmacniacz dźwięku, 105

X

XBee, 305, 310, 315
  Explorer, 311

Z

zalanie piwnicy, 408
zamek z klawiaturą numeryczną, 221
zapisywanie
  szkicu, 43
  zmodyfikowanego szkicu, 118
zasilanie, 39
zatrzask, 141

zdalne sterowanie robota, 325
zegar, 141
czasu rzeczywistego, 361
ustawianie daty i godziny, 362
precyzyjny, 296
zewnętrzna pamięć EEPROM, 346
zintegrowane
środowisko programowania, 25, 42
kontroler sieciowy, 380
złącza, 39
zmienna, 64
logiczna, 90
tablicowa, 389
wyświetlanie wartości, 122
znakowy moduł LCD, 173

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

http://program-partnerski.helion.pl

GRUPA WYDAWNICZA
Helion SA